



HZ BOOKS
华章科技

机器学习领域经典著作，智能计算专家多年经验结晶，以全新的角度诠释机器学习的算法理论，
通过案例系统阐述机器学习的实践方法和应用技巧，指导读者轻松步入工程应用阶段



技术丛书



The Practice of Machine Learning

机器学习实践指南

案例应用解析

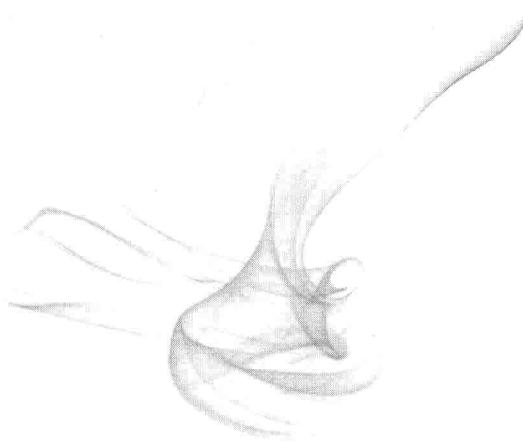
麦好◎著



The Practice of Machine Learning

机器学习实践指南 案例应用解析

麦好◎著



机械工业出版社
China Machine Press

www.aibbt.com 让未来触手可及

图书在版编目 (CIP) 数据

机器学习实践指南：案例应用解析/麦好著. —北京：机械工业出版社，2014.4
(大数据技术丛书)

ISBN 978-7-111-46207-1

I. 机… II. 麦… III. 机器学习—指南 IV. TP181-62

中国版本图书馆CIP数据核字 (2014) 第054810号

本书是机器学习及数据分析领域不可多得的一本著作，也是为数不多的既有大量实践应用案例又包含算法理论剖析的著作，作者针对机器学习算法既抽象复杂又涉及多门数学学科的特点，力求理论联系实际，始终以算法应用为主线，由浅入深以全新的角度诠释机器学习。

全书分为准备篇、基础篇、统计分析实战篇和机器学习实战篇。准备篇介绍了机器学习的发展及应用前景以及常用科学计算平台，主要包括统计分析语言 R、机器学习模块 mipy 和 Neurolab、科学计算平台 Numpy、图像识别软件包 OpenCV、网页分析 BeautifulSoup 等软件的安装与配置。基础篇先对数学基础及其在机器学习领域的应用进行讲述，同时推荐配套学习的数学书籍，然后运用实例说明计算平台的使用，以 Python 和 R 为实现语言，重点讲解了图像算法、信息隐藏、最小二乘法拟合、因子频率分析、欧氏距离等，告诉读者如何使用计算平台完成工程应用。最后，通过大量统计分析和机器学习案例提供实践指南，首先讲解回归分析、区间分布、数据图形化、分布趋势、正态分布、分布拟合等数据分析基础，然后讲解神经网络、统计算法、欧氏距离、余弦相似度、线性与非线性回归、数据拟合、线性滤波、图像识别、人脸识别、网页分类等机器学习算法。此书可供算法工程师、IT 专业人员以及机器学习爱好者参考使用。

机器学习实践指南：案例应用解析

麦好 著

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）

责任编辑：陈佳媛 杨绣国

印 刷：冀城市京瑞印刷有限公司

版 次：2014年4月第1版第1次印刷

开 本：186mm×240mm 1/16

印 张：21.25（含0.25印张彩插）

书 号：ISBN 978-7-111-46207-1

定 价：69.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光/邹晓东

华章计算机
HZBOOKS | Computer Science and Technology



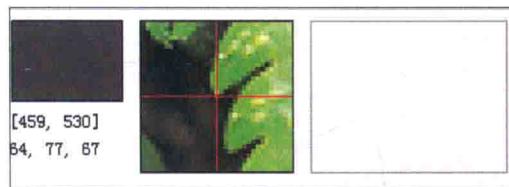


图 4-6 树叶放大的颗粒效果



图 4-9 随机产生若干像素点

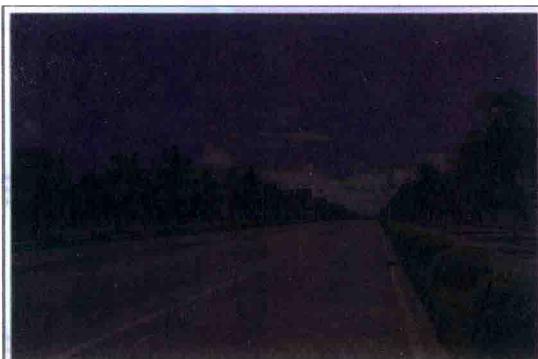


图 4-10 图像变暗

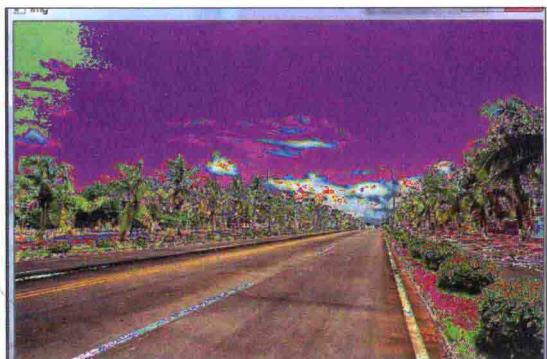


图 4-11 图像变亮

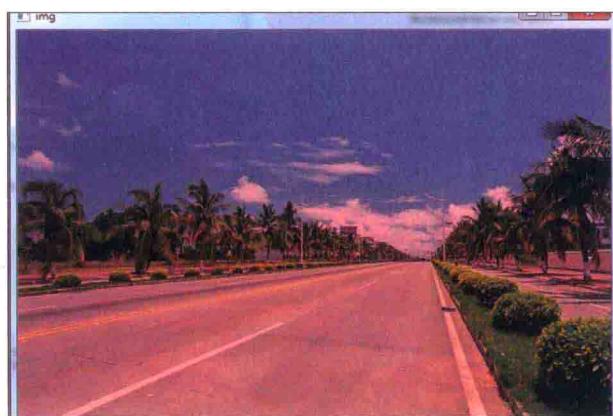


图 4-12 图像日落效果

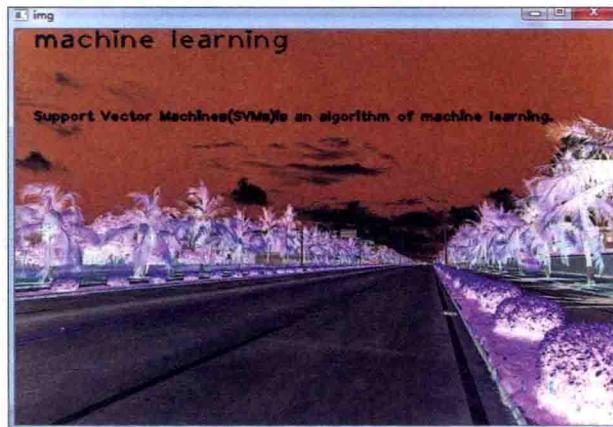


图 4-13 负片和水印效果

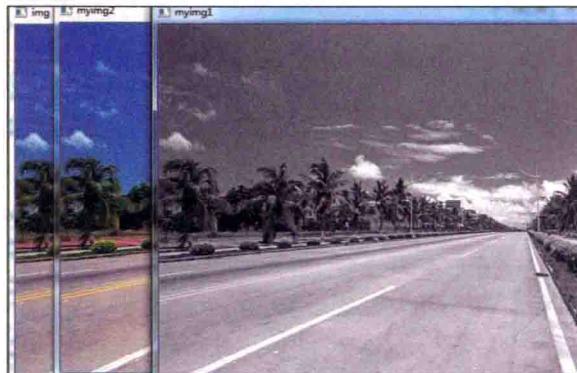


图 4-18 图像灰度化

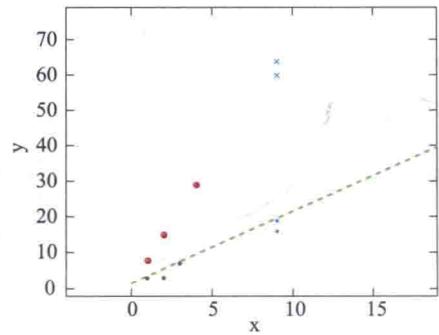


图 7-4 神经网络分类

Gradient Descent Algorithm

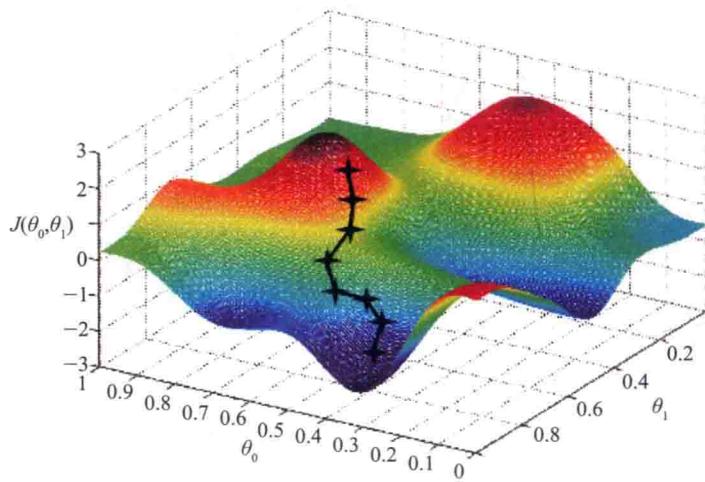


图 7-6 误差曲面及梯度下降

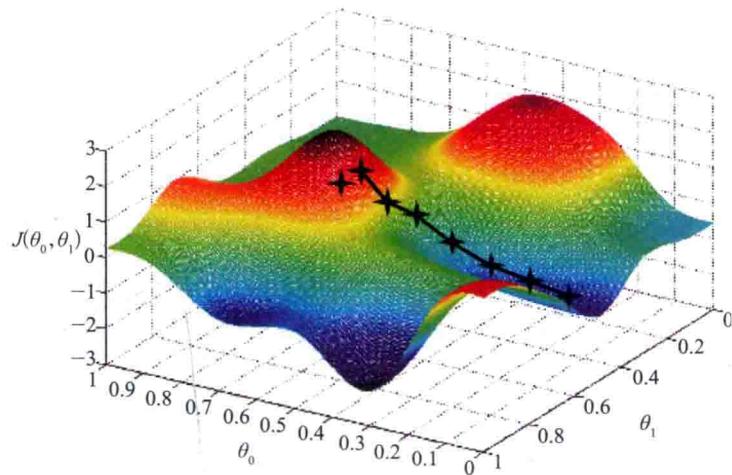


图 7-7 落到局部最小点的梯度下降

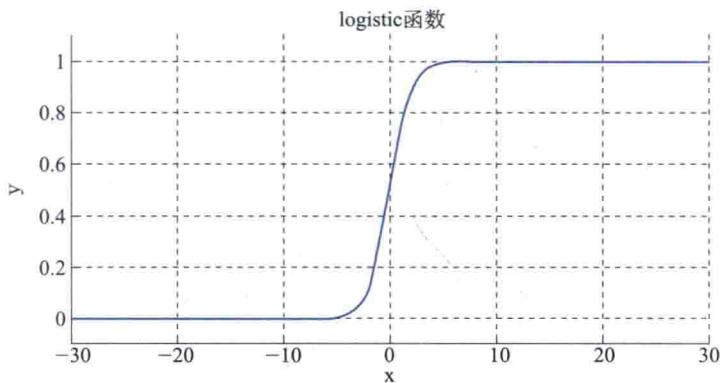


图 7-12 sigmoid 曲线

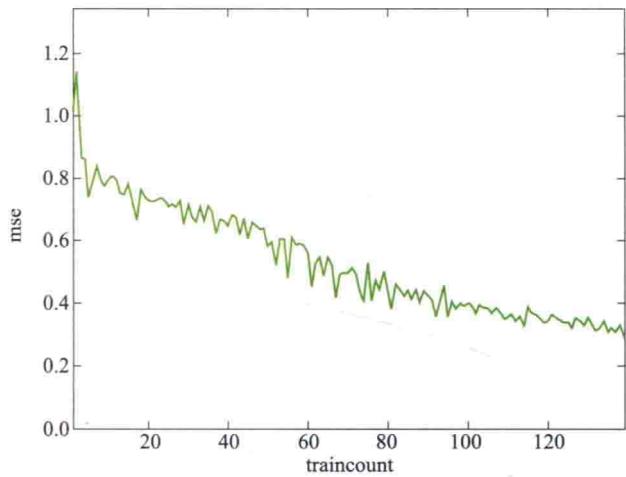


图 7-18 误差曲线

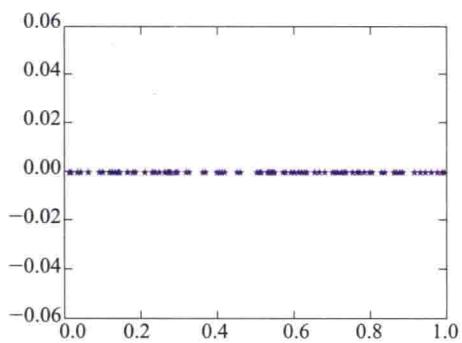


图 7-23 数据点分布



图 9-5 弱噪声切片识别效果图

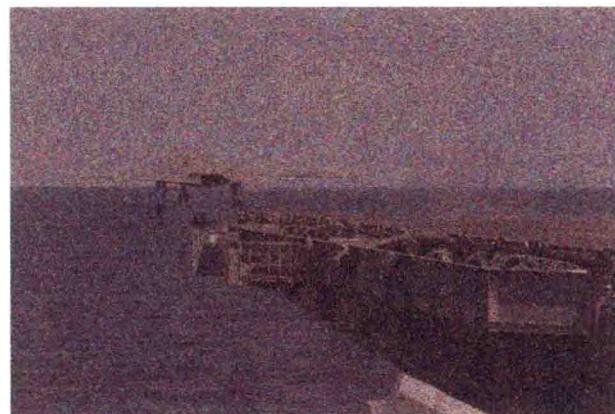


图 9-6 强噪声图像



图 9-7 强噪声切片识别效果图
www.aibbt.com 让未来触手可及

前　　言

为什么要写这本书

自从计算机问世以来，人们就想知道，机器是否能像人类一样具有学习能力。1959年，美国的塞缪尔设计了一个下棋程序，这个程序具有学习能力，它可以在不断的对弈中提高自己的棋艺。4年后，这个程序战胜了设计者本人。又过了3年，这个程序战胜了美国一个保持常胜不败战绩8年之久的冠军。不难看出，这个程序向人们展示了机器学习的能力。如果我们理解了计算机学习的内在机制，即怎样使它们根据经验来自动提高，那么影响将是空前的。

机器学习作为一门多领域交叉学科，在近20年里异军突起。机器学习涉及概率论、统计学、代数学、微积分、算法复杂度理论等多门学科，通过设计和分析一些让计算机可以自动“学习”的算法，人类对机器学习的不断研究开辟出许多全新的应用领域，使智能机器的计算能力和可定制性上升到新的高度。

在国外，机器学习技术大量应用于军事领域，X-47B 验证机已经完成首飞，这款由诺斯罗普·格鲁曼公司为美国海军研制、外形极似B-2 战略轰炸机的飞机，是世界上第一架完全由计算机控制的“无尾翼、喷气式无人驾驶飞机”，它意味着在未来战场，将会出现无人机先出动，打击对方的防空阵地、雷达、机场等重要目标，而有人机编队则在战场外，负责拦截对方空中支援的战斗机，这将彻底改变人类战争的方式。X-47B代表了人类在机器学习研究方面的巨大进步，是智能机器全面参与人类战争的标志，代表了人类在模仿自己智能水平方面进入了一个新的阶段，同时也给机器学习带来了全新的发展机会。

在国内，机器学习正展现出巨大的潜力，在计算机领域中扮演着日益重要的角色。机器学习的应用领域包括数据挖掘、语音识别、图像识别、机器人、生物信息学、信息安全、车辆自动驾驶、遥感信息处理、计算金融学、工业过程控制、智能家居等。在不久的将来，机器的学习能力更接近人类智能：计算机能通过学习医疗记录，获取治疗新疾病最有效的方法；住宅管理系统可分析住户的用电模式，以降低能源消耗；个人助理软件则可跟踪用户的兴趣，为其选择最感兴趣的在线信息。

随着机器学习技术在国内外的大量应用，机器学习工程师成了备受关注的人才。Google、Microsoft等公司早已经尝到了机器学习商业化带来的甜头，所以对机器学习人才提出了大量的需

求。国内很多知名的公司如阿里巴巴、淘宝等为迎接大数据时代带来的挑战，已经大量引进机器学习方面的人才。百度、搜狗等由于拥有能与Google竞争的搜索引擎，早就开始了机器学习人才的猎取。奇虎作为中国领先的互联网软件与技术公司，其重头产品360安全卫士成为网络安全领域的领先品牌，也对引进机器学习研发工程师表现出了强烈的渴求。

现在中国已经悄然兴起了机器学习的学习热潮，掌握机器学习的工程师成为了各大IT巨头手中疯抢的“香饽饽”。机器学习成为了进入国内知名IT公司和跨国IT巨头比如Microsoft、Google的敲门砖，良好的发展势头和较高的职业薪水，吸引着越来越多的软件工程师和数据分析师涌人机器学习领域。

但是，机器学习的入门门槛较高，尤其对研究者的数学理解能力有较高要求，相对于数据结构、计算机算法以及系统架构知识来说，机器学习是一个全新的领域，也是一个全新的高度。希望本书能帮助读者进入机器学习的精彩世界。

理解机器学习算法往往要从理解它涉及的数学公式和数学知识开始，本书作者也是通过攀登数学这座大山一步步走入机器学习领域的，对此深有体会。打好数学基础是非常必要的，一旦你掌握了数学分析、线性代数、概率与统计、统计学、离散数学、抽象代数、数学建模等数学理论后，理解机器学习算法就容易多了，就不会畏惧那些让人生厌和烦琐的数学符号和数学公式，反而会喜欢上这些数学公式，并尝试亲自推导一番。

读者对象

- 开发人员。在理解机器学习算法的基础上，调用机器学习的中间库进行开发，将机器学习应用于各种场景，如数据分析、图像识别、文本分类、搜索引擎、中文智能输入法等。
- 架构师。在理解机器学习算法的基础上，适应现代云计算平台的发展，将机器学习算法应用在大规模并行计算上。同时，机器学习算法是大数据分析的基础，如神经网络、SVM、相似度分析、统计分析等技术。
- 机器学习的初、中级读者。人类对机器学习的研究只是一个开始，还远远没有结束。近年来，机器学习一直保持着强劲的发展势头，并拥有广阔的发展前景，而不同于某些软件开发领域中的程序语言或架构知识。掌握机器学习有一定的难度，属于“金领”行业，对读者来说，掌握机器学习知识就意味着更高的薪水、更具前景的职业。

如何阅读本书

全书分为准备篇、基础篇、统计分析实战篇和机器学习实战篇。机器学习算法建立在复杂的计算理论基础之上，并涉及多门数学学科。抽象的理论加上成堆的数学公式，对部分读者来说，带来了极大的挑战，也许会将渴求学习的人们挡在门外。针对这种情况，本书力求理论联系实际，在介绍理论基础的同时，注重机器学习算法的实际运用，让读者明白其中的原理。

准备篇中首先介绍机器学习的发展及应用前景，使读者对其产生深厚的兴趣，同时也介绍目前

常用的科学计算平台和本书将用到的工程计算平台，使读者消除对机器学习的畏难心理。这些平台的使用，也降低了机器学习软件实现的难度。

基础篇将对数学知识基础、计算平台应用实例进行介绍，推荐配置学习的数学教科文档，介绍计算平台开发的基本知识，应用这些平台实现计算应用。

最后，本书将针对统计分析实战和机器学习实战两个部分帮助读者建立机器学习实战指南。还将大量应用计算平台对统计分析以及机器学习算法，并进行软件的实现和应用。本书附有效果图，使读者对机器学习的应用和理论基础有形象的理解。

勘误和支持

由于作者的水平有限，编写的时间也很仓促，书中难免会出现一些错误或者不准确的地方，有不妥之处恳请读者批评指正。您如果遇到任何问题，或有更多的宝贵意见，欢迎发送邮件至我的邮箱myhaspl@myhaspl.com，很期待能够收到您的真挚反馈。此外，本书的代码及相关资源请在华章网站（<http://www.hzbook.com/>）本书页面上下载。

致谢

我首先要感谢伟大的电影《机械公敌》及其主角威尔·史密斯，这位美国演员主演了《当幸福来敲门》、《拳王阿里》、《绝地战警》、《全民超人汉考克》、《黑衣人》、《机械公敌》等影片，他曾获奥斯卡奖和金球奖提名。他主演的《当幸福来敲门》让很多人理解到了幸福是什么，而《机械公敌》让我看到了人工智能的未来，我相信《机械公敌》描述的以下场景一定能在将来实现：

公元2035年，智能型机器人已被人类广泛利用，作为最好的生产工具和人类伙伴，机器人在各个领域扮演着日益重要的角色。而由于有众所周知的机器人“三大安全法则”的限制，人类对这些能够胜任各种工作且毫无怨言的伙伴充满信任，它们中的很多甚至已经成为了一个家庭的组成成员。

但是我不希望看到电影中的NS-5型机器人追杀和控制人类的场景在将来某一天上演，这将是人类的悲剧，我想这并不是人工智能学者希望看到的。也许将来有一天，人工智能技术很成熟了，机器人与人之间的关系可以作为一个社会伦理和哲学问题被大家热议，机器人也能和人类一起参与讨论自己在人类社会中的角色和定位。

我衷心感谢机械工业出版社华章公司的编辑们，由于他们的努力和远见，让我顺利地完成了全部书稿。最后我还要感谢家人的大力支持和无私奉献，正因为有他们的关心和照顾，我才有足够的时间和精力来完成本书的撰写工作。

谨以此书献给热爱机器学习技术的朋友以及喜欢威尔·史密斯的影迷。

麦好（Myhaspl）

中国，广东，2013年12月

目 录

前 言

第一部分 准备篇

第1章 机器学习发展及应用前景	2
1.1 机器学习概述	2
1.1.1 什么是机器学习	3
1.1.2 机器学习的发展	3
1.1.3 机器学习的未来	4
1.2 机器学习应用前景	5
1.2.1 数据分析与挖掘	5
1.2.2 模式识别	5
1.2.3 更广阔的领域	6
1.3 小结	7
第2章 科学计算平台	8
2.1 科学计算软件平台概述	8
2.1.1 常用的科学计算软件	9
2.1.2 本书使用的工程计算平台	10
2.2 计算平台的配置	11
2.2.1 Numpy等Python科学计算包的安装与配置	11
2.2.2 OpenCV 安装与配置	13

2.2.3 mlp	安装与配置	14
2.2.4 BeautifulSoup	安装与配置	15
2.2.5 Neurolab	安装与配置	15
2.2.6 R	安装与配置	15
2.3 小结	16

第二部分 基础篇

第3章 机器学习数学基础	18
3.1 数学对我们有用吗	18
3.2 机器学习需要哪些数学知识	20
3.3 小结	25
第4章 计算平台应用实例	26
4.1 Python计算平台简介及应用实例	26
4.1.1 Python语言基础	26
4.1.2 Numpy库	37
4.1.3 pylab、matplotlib绘图	44
4.1.4 图像基础	46
4.1.5 图像融合与图像镜像	55
4.1.6 图像灰度化与图像加噪	57
4.1.7 声音基础	60
4.1.8 声音音量调节	63
4.1.9 图像信息隐藏	68
4.1.10 声音信息隐藏	72
4.2 R语言基础	78
4.2.1 基本操作	78
4.2.2 向量	81
4.2.3 对象集属性	87
4.2.4 因子和有序因子	88
4.2.5 循环语句	89

4.2.6 条件语句	89
4.3 R语言科学计算.....	90
4.3.1 分类(组)统计	90
4.3.2 数组与矩阵基础	91
4.3.3 数组运算	94
4.3.4 矩阵运算	95
4.4 R语言计算实例.....	103
4.4.1 学生数据集读写	103
4.4.2 最小二乘法拟合	105
4.4.3 交叉因子频率分析	106
4.4.4 向量模长计算	107
4.4.5 欧氏距离计算	108
4.5 小结	109
思考题	109

第三部分 统计分析实战篇

第5章 统计分析基础	112
5.1 数据分析概述	112
5.2 数学基础	113
5.3 回归分析	118
5.3.1 单变量线性回归	118
5.3.2 多元线性回归	121
5.3.3 非线性回归	121
5.4 数据分析基础	124
5.4.1 区间频率分布	124
5.4.2 数据直方图	126
5.4.3 数据散点图	127
5.4.4 五分位数	129
5.4.5 累积分布函数	130
5.4.6 核密度估计	130

5.5 数据分布分析	132
5.6 小结	134
思考题	135
第6章 统计分析案例	136
6.1 数据图形化案例解析	136
6.1.1 点图	136
6.1.2 饼图和条形图	137
6.1.3 茎叶图和箱线图	138
6.2 数据分布趋势案例解析	140
6.2.1 平均值	140
6.2.2 加权平均值	140
6.2.3 数据排序	141
6.2.4 中位数	142
6.2.5 极差、半极差	142
6.2.6 方差	143
6.2.7 标准差	143
6.2.8 变异系数、样本平方和	143
6.2.9 偏度系数、峰度系数	144
6.3 正态分布案例解析	145
6.3.1 正态分布函数	145
6.3.2 峰度系数分析	146
6.3.3 累积分布概率	146
6.3.4 概率密度函数	147
6.3.5 分位点	148
6.3.6 频率直方图	151
6.3.7 核概率密度与正态概率分布图	151
6.3.8 正太检验与分布拟合	152
6.3.9 其他分布及其拟合	154
6.4 小结	155
思考题	155

第四部分 机器学习实战篇

第7章 机器学习算法	158
7.1 神经网络	158
7.1.1 Rosenblatt感知器	159
7.1.2 梯度下降	173
7.1.3 反向传播与多层感知器	180
7.1.4 Python神经网络库	199
7.2 统计算法	201
7.2.1 平均值	201
7.2.2 方差与标准差	203
7.2.3 贝叶斯算法	205
7.3 欧氏距离	208
7.4 余弦相似度	209
7.5 SVM	210
7.5.1 数学原理	210
7.5.2 SMO算法	212
7.5.3 算法应用	212
7.6 回归算法	217
7.6.1 线性代数基础	217
7.6.2 最小二乘法原理	218
7.6.3 线性回归	219
7.6.4 多元非线性回归	221
7.6.5 岭回归方法	223
7.6.6 伪逆方法	224
7.7 PCA降维	225
7.8 小结	227
思考题	227
第8章 数据拟合案例	228
8.1 数据拟合	228
8.1.1 图像分析法	228

8.1.2 神经网络拟合法	240
8.2 线性滤波	256
8.2.1 WAV声音文件	256
8.2.2 线性滤波算法过程	256
8.2.3 滤波Python实现	257
8.3 小结	262
思考题	262
第9章 图像识别案例	264
9.1 图像边缘算法	264
9.1.1 数字图像基础	264
9.1.2 算法描述	265
9.2 图像匹配	266
9.2.1 差分矩阵求和	267
9.2.2 差分矩阵均值	269
9.2.3 欧氏距离匹配	271
9.3 图像分类	277
9.3.1 余弦相似度	277
9.3.2 PCA图像特征提取算法	283
9.3.3 基于神经网络的图像分类	284
9.3.4 基于SVM的图像分类	289
9.4 人脸辨识	291
9.4.1 人脸定位	291
9.4.2 人脸辨识	293
9.5 手写数字识别	300
9.5.1 手写数字识别算法	300
9.5.2 算法的Python实现	301
9.6 小结	303
思考题	304
第10章 文本分类案例	305
10.1 文本分类概述	305

10.2 余弦相似度分类	306
10.2.1 中文分词	306
10.2.2 停用词清理	308
10.2.3 算法实战	310
10.3 朴素贝叶斯分类	315
10.3.1 算法描述	316
10.3.2 先验概率计算	316
10.3.3 最大后验概率	316
10.3.4 算法实现	317
10.4 小结	323
思考题	323

第一部分

准备篇

子贡问为仁。子曰：“工欲善其事，必先利其器。居是邦也，事其大夫之贤者，友其士之仁者。”

——孔子

第1章

机器学习发展及应用前景

纵观国内软件工程师的发展路线，前期多以程序员（“码农”）、测试工程师、数据库管理员、多媒体技术员、网页与信息技术员等职业为主；中期主要是软件设计师、软件评测师、技术支持师等职业；后期是职业发展的黄金阶段，这一阶段对于拥有丰富技术经验的工程师来说十分重要，但这一阶段容易遭遇到技术发展瓶颈，因此，很多人将目光投向了项目管理和系统架构，比如：系统架构师、项目管理师等。

近年来，国内对机器学习的研究日益深入，应用领域不断扩大，催生了新的IT职位——机器学习工程师，百度、搜狗、阿里巴巴、淘宝、奇虎等国内IT巨头纷纷提出了对机器学习工程师的需求，掌握机器学习的人才成为了各大IT厂商争抢的“香饽饽”。机器学习迅速走红成为热门技术，这给软件工程师带来了绝佳的发展机遇，研究与应用机器学习算法成为了突破技术瓶颈的方式，机器学习工程师、项目管理师和系统架构师并称为后期发展的三大黄金职位。

1.1 机器学习概述

机器学习作为一门多领域的交叉学科，在近20年里异军突出。机器学习涉及概率论、统计学、微积分、代数学、算法复杂度理论等多门学科。通过可以让计算机自动“学习”的算法来实现人工智能，是人类在人工智能领域展开的积极探索。

2009年，被誉为人工大脑之父的雨果·德·加里斯教授走进清华大学讲堂，在两小时的演讲时间内，给大家描述了一个人工智能的世界：20年后，人工智能机器可以和人类做朋友，50年后，人工智能将成为人类最大的威胁，世界最终会因人工智能超过人类而爆发一场战争，这场智能战争也许会夺去数十亿人的生命。这样的描述并不是幻想，随着人类在人工智能领域取得的进步，这很有可能成为事实。而这一切主要归功于对机器学习的研究和探索。

1.1.1 什么是机器学习

学习是人类具有的一种重要智能行为。人类一直梦想机器能像人类一样学习，也一直在为这个终极目标努力。那么，什么是机器学习呢？长期以来众说纷纭，Langley (1996) 定义机器学习为：“机器学习是一门人工智能的科学，该领域的研究对象是人工智能，特别是如何在经验学习中改善具体算法的性能” (Machine learning is a science of the artificial. The field's main objects of study are artifacts, specifically algorithms that improve their performance with experience.)。Mitchell (1997) 在《Machine Learning》中写到：“机器学习是计算机算法的研究，并通过经验提高其自动进行改善” (Machine Learning is the study of computer algorithms that improve automatically through experience.)。Alpaydin (2004) 提出自己对机器学习的定义：“机器学习是用数据或以往的经验，来优化计算机程序的性能标准” (Machine learning is programming computers to optimize a performance criterion using example data or past experience.)。

笔者综合维基百科和百度百科的定义，尝试着将机器学习定义如下：“机器学习是一门人工智能的科学，该领域的研究对象是人工智能，专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能，它是人工智能的核心，是使计算机具有智能的根本途径。机器学习的研究方法通常是根据生理学、认知科学等对人类学习机理的了解，建立人类学习过程的计算模型或认识模型，发展各种学习理论和学习方法，研究通用的学习算法并进行理论上的分析，建立面向任务的具有特定应用的学习系统。”

1.1.2 机器学习的发展

早在古代，人类就萌生了制造出智能机器的想法。中国人在 4500 年前发明的指南车，以及三国时期诸葛亮发明的尽人皆知的木牛流马；日本人在几百年前制造过靠机械装置驱动的玩偶；1770 年英国公使给中国皇帝进贡了一个能写“八方向化，九土来王” 8 个汉字的机器玩偶（这个机器人至今还保存在故宫博物院），等等。这些例子，都只是人类早期对机器学习的一种认识和尝试。

真正的机器学习研究起步较晚，它的发展过程大体上可分为以下 4 个时期：

第一阶段是在 20 世纪 50 年代中叶到 20 世纪 60 年代中叶，属于热烈时期。

第二阶段是在 20 世纪 60 年代中叶至 20 世纪 70 年代中叶，被称为机器学习冷静期。

第三阶段是从 20 世纪 70 年代中叶至 20 世纪 80 年代中叶，称为机器学习复兴期。

最新的阶段起始于 1986 年。当时，机器学习综合应用了心理学、生物学和神经生理学以及数学、自动化和计算机科学，并形成了机器学习理论基础，同时还结合各种学习方法取长补短，形成集成学习系统。此外，机器学习与人工智能各种基础问题的统一性观点正在形成，各种学习方法的应用范围不断扩大，同时出现了商业化的机器学习产品，还积极开展了与机器学习有关的学术活动。

1989 年, Carbonell 指出机器学习有 4 个研究方向: 连接机器学习、基于符号的归纳机器学习、遗传机器学习与分析机器学习。1997 年, Dietterich 再次提出了另外 4 个新的研究方向: 分类器的集成 (Ensembles of classifiers)、海量数据的有教师学习算法 (Methods for scaling up supervised learning algorithm)、增强机器学习 (Reinforcement learning) 与学习复杂统计模型 (Learning complex stochastic models)。

在机器学习的发展道路上, 值得一提的是世界人工大脑之父雨果·德·加里斯教授。他创造的 CBM 大脑制造机器可以在几秒钟内进化成一个神经网络, 可以处理将近 1 亿个人工神经元, 它的计算能力相当于 10000 台个人计算机。在 2000 年, 人工大脑可以控制“小猫机器人”的数百个行为能力。

2010 年以来, Google、Microsoft 等国际 IT 巨头加快了对机器学习的研究, 已经尝到了机器学习商业化带来的甜头, 国内很多知名的公司也纷纷效仿。阿里巴巴、淘宝为应付大数据时代带来的挑战, 已经在自己的产品中大量应用机器学习算法。百度、搜狗等已拥有能与 Google 竞争的搜索引擎, 其产品中也早已融合了机器学习知识, 360 安全卫士的奇虎公司也意识到了机器学习的意义所在, 这些大公司纷纷表现出对机器学习研发工程师的渴求。近几年正是机器学习知识在国内软件工程师群体中普及的黄金时代, 也给软件工程师们进入机器学习这一金领行业带来了机遇。

1.1.3 机器学习的未来

展望未来, 出现在《终结者》等系列电影上的场景终将成为现实, 并将在未来的人类世界中频频上演。

目前人类已经进入了与高智能机器共同参与战争的新时代。据美国《航空周刊与空间技术》报道, X-47B 验证机已经完成首飞。这款由诺斯罗普·格鲁曼公司为美国海军研制、外形极似 B-2 战略轰炸机的飞机, 是世界上第一架完全由计算机控制的“无尾翼、喷气式无人驾驶飞机”, 它意味着在未来的海空战场, 将会出现无人机先出动, 打击对方的防空阵地、雷达、机场等重要目标, 而有人机编队则在战场外, 负责拦截对方空中支援的战斗机的作战模式。这将彻底改变人类战争的方式。未来人类在机器学习研究领域的发展将会进一步推动机器人军队在战场上的应用。

此外, 智能机器已经深入到人类的生活、工作中。在民用领域, 能从医疗记录中学习的机器将会出现, 它们能分析和获取治疗新疾病最有效的方法; 智能家居高度发展, 分析住户的用电模式、居住习惯后, 打造动态家居, 从而降低能源消耗、提高居住舒适度; 个人智能助理跟踪分析用户的职业和生活细节, 协助用户高效完成工作和享受健康生活。所有这些都将有智能机器的功劳。

不久的将来, 人类也许该思考: 在未来的世界里, 机器人将充当什么样的角色, 会不会代替人类呢? 人类与智能机器之间应如何相处?

人类开始着手研究, 如何才能更好地实现下面三大准则:

第一, 机器人不可伤害人;

第二，机器人必须服从人的命令；

第三，机器人可以在不违背上述原则的情况下保护自己。

1.2 机器学习应用前景

机器学习应用广泛，无论是在军事领域还是民用领域，都有机器学习算法施展的机会。

1.2.1 数据分析与挖掘

“数据挖掘”和“数据分析”通常被相提并论，并在许多场合被认为是可以相互替代的术语。关于数据挖掘，现在已有多种文字不同但含义接近的定义，例如“识别出巨量数据中有效的、新颖的、潜在有用的、最终可理解的模式的非平凡过程”；百度百科将数据分析定义为：“数据分析是指用适当的统计方法对收集来的大量第一手资料和第二手资料进行分析，以求最大化地开发数据资料的功能，发挥数据的作用，它是为了提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。”无论是数据分析还是数据挖掘，都是帮助人们收集、分析数据，使之成为信息，并作出判断，因此可以将这两项合称为“数据分析与挖掘”。

数据分析与挖掘技术是机器学习算法和数据存取技术的结合，利用机器学习提供的统计分析、知识发现等手段分析海量数据，同时利用数据存取机制实现数据的高效读写。机器学习在数据分析与挖掘领域中拥有无可取代的地位，2012年Hadoop进军机器学习领域就是一个很好的例子。

2012年，Cloudera收购Myrrix共创Big Learning，从此，机器学习俱乐部多了一名新会员。Hadoop和便宜的硬件使得大数据分析更加容易，随着硬盘和CPU越来越便宜，以及开源数据库和计算框架的成熟，创业公司甚至个人都可以进行TB级以上的复杂计算。Myrrix从Apache Mahout项目演变而来，是一个基于机器学习的实时可扩展的集群和推荐系统。

Myrrix创始人Owen在其文章中提到：机器学习已经是一个有几十年历史的领域了，为什么大家现在这么热衷于这项技术？因为大数据环境下，更多的数据使机器学习算法表现得更好，机器学习算法能从数据海洋提取更多有用的信息；Hadoop使收集和分析数据的成本降低，学习的价值提高。Myrrix与Hadoop的结合是机器学习、分布式计算和数据分析与挖掘的联姻，这三大技术的结合让机器学习应用场景呈爆炸式的增长，这对机器学习来说是一个千载难逢的好机会。

1.2.2 模式识别

模式识别起源于工程领域，而机器学习起源于计算机科学，这两个不同学科的结合带来了模式识别领域的调整和发展。模式识别研究主要集中在两个方面：一是研究生物体（包括人）是如何感知对象的，属于认识科学的范畴；二是在给定的任务下，如何用计算机实现模式识别的理论和方法，这些是机器学习的长项，也是机器学习研究的内容之一。

模式识别的应用领域广泛，包括计算机视觉、医学图像分析、光学文字识别、自然语言处理、语音识别、手写识别、生物特征识别、文件分类、搜索引擎等，而这些领域也正是机器学习的大展身手的舞台，因此模式识别与机器学习的关系越来越密切，以至于国外很多书籍把模式识别与机器学习综合在一本里讲述。

1.2.3 更广阔领域的领域

目前国外的 IT 巨头正在深入研究和应用机器学习，他们把目标定位于全面模仿人类大脑，试图创造出拥有人类智慧的机器大脑。

2012 年 Google 在人工智能领域发布了一个划时代的产品——人脑模拟软件，这个软件具备自我学习功能，模拟脑细胞的相互交流，可以通过看 YouTube 视频学习识别猫、人以及其他事物。当有数据被送达这个神经网络的时候，不同神经元之间的关系就会发生改变。而这也使得神经网络能够得到对某些特定数据的反应机制，据悉这个网络现在已经学到了一些东西，Google 将有望在多个领域使用这一新技术，最先获益的可能是语音识别。

与此同时，Google 研制的自动驾驶汽车于 2012 年 5 月获得了美国首个自动驾驶车辆许可证，将于 2015 年至 2017 年进入市场销售，如图 1-1 所示。

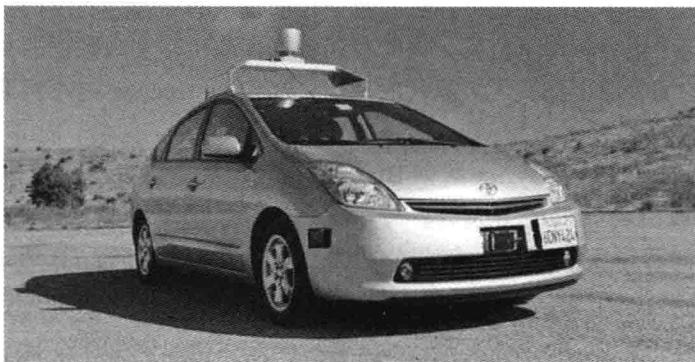


图 1-1 Google 研制的自动驾驶汽车

自动驾驶汽车依靠人工智能、视觉计算、雷达、监控装置和全球定位系统协同合作，让电脑可以在没有任何人类主动操作的情况下，通过计算机自动安全地操作机动车辆，Google 认为：这将是一种“比人更聪明的”汽车，不仅能预防交通事故，还能节省行驶时间、降低碳排放量。

2013 年，Microsoft CEO 高级顾问 Craig Mundie 在北京航空航天大学学术交流厅发表“科技改变未来”的主题演讲，Mundie 在演讲中谈到了当今 IT 科技的三大挑战：大数据、人工智能和人机互动。他认为随着大数据时代的到来，人们的各种互动、设备、社交网络和传感器正在生成海量的数据，而机器学习可以更好地处理这些数据，挖掘其中的潜在价值。与此同时，他展示了微软研究院在机器学习方面的新产品——英语转汉语实时拟原声翻译，研究过计算语言学的朋友都知道自然语言理解与处理属于机器学习的问题，让计算机理解人

类语言可以视同创造出一个机器，该机器拥有与人类一样聪明的智慧。

机器学习在军事上的应用更加广泛，智能无人机、智能无人舰艇、智能无人潜艇陆续研究成功或已投放战场，其他军事领域也有机器学习研究成果的应用，如：美国国防部高级研究计划局的电子战专家正在尝试推出利用机器学习技术对抗敌方的无线自适应通信威胁，其发布了一份概括性机构通告（DARPA-BAA-10-79），内容为“自适应电子战行为学习”计划（BLADE），以研发确保美国电子战系统能够在战场上学习自动干扰新式射频威胁的算法和技术。

1.3 小结

机器学习作为一门多领域交叉学科，该领域的主要研究对象是人工智能，专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构，使之不断改善自身的性能，它是人工智能的核心，是使计算机具有智能的根本途径。

近年来，机器学习的研究与应用在国内外越来越重视。机器学习已经广泛应用于语音识别、图像识别、数据挖掘等领域。大数据时代的到来，使机器学习有了新的应用领域，从包含设备维护、借贷申请、金融交易、医疗记录、广告点击、用户消费、客户网络行为等数据中发现有价值的信息已经成为其研究与应用的热点。

我们以记者与雨果·德·加里斯教授的部分专访内容来结束这一章。

记者：为什么选择这个研究工作？

雨果·德·加里斯：对人类大脑的好奇心和人脑的想象力的好奇心。人类只是一个个分子构成的机器，像计算机的芯片一样，像编制程序一样。另一方面，作为生物人都会死亡消失的，但人工智能机器就不会。所以说，这个研究就像制造神一样。

当人类促使技术进步，让具有人工智能的机器人得以诞生和发展，但总有一天人工智能机器会实现自己进化，当这种技术达到一个奇点的时候，就不需要人类来推动了。比人类聪明得多的人工智能机器将在以年为单位的短时间里产生。

记者：预测一下未来人工智能机器的前景。

雨果·德·加里斯：下一个20年，它们很有可能出现在我们的家里，为我们打扫房间，照顾小孩，和我们聊天，给我们来自地球上知识库里面的无限知识。我们还将可以和它们有性关系，被它们教育，从它们那里得到娱乐和开怀大笑。20年后的大脑制造业，每年全球范围内将可能创造万亿美元的价值。人工智能机器有比我们聪明万亿倍的可能性，不夸张地说，人工智能机器和人类交流，就像人类试图和岩石交流一样艰难。不过，真正的人工智能在我死后的三四十年内不会被制造出来。我活着看不到工作的真正结果，这是让我沮丧和失望的一个根源。

第2章

科学计算平台

机器学习算法具有坚实的数学理论支持，机器学习的应用建立在科学计算的基础上，而数学计算又是科学计算的主要组成部分。计算机技术的飞速发展和计算数学方法及理论的日益成熟，使解决复杂的数学计算问题成为可能。这些问题在以前用一般的计算工具来解决非常困难，而现在用计算机来处理却非常容易。目前用计算机处理得较多的数学计算主要分为以下两类：

第一类是数值计算，它以数值数组作为运算对象，给出数值解；计算过程中可能会产生误差累积问题，影响了计算结果的精确性；计算速度快，占用资源少。

第二类是符号计算，它以符号对象和符号表达式作为运算对象，给出解析解；运算不受计算误差累积问题的影响；计算指令简单；占用资源多，计算耗时长。

数值计算方法成为了科学计算的重要手段，它研究怎样利用计算工具来求出数学问题的数值解。数值计算方法的计算对象是微积分、线性代数、插值与逼近及最小二乘拟合、数值积分与数值微分、矩阵的特征值与特征向量求解、线性方程组与非线性方程求根，以及微分方程数值解法等数学问题，这些是模式识别、数据分析及自动制造等机器学习领域需要应用的数学。

符号计算是专家系统等机器学习领域需要应用的数学，在符号计算中，计算机处理的数据和得到的结果都是符号。符号既可以是字母和公式，也可以是数值，其运算以推理解析的方式进行，不受计算误差积累问题困扰，计算结果为完全正确的封闭解或任意精度的数值解，这意味着符号计算给出的结果能避免因舍入误差而引起的问题。还有更多的数学分支正在进入机器学习领域，复杂的数学计算需要强大的科学计算平台。科学计算平台提供了机器学习算法应用的底层支持。

2.1 科学计算软件平台概述

现代科学研究的方法主要有三种：理论论证、科学实验、科学计算。近年来，科学计算

方法逐步成为科学的研究的主流方法，在金融工程、信息检索、基因研究、环境模拟、数值计算、数据分析、决策支持等领域得到了广泛使用。由于计算机技术的发展及其在各技术科学领域的应用推广与深化，这些应用领域不论其背景与含义如何，都要用计算机进行科学计算，都必须建立相应的数学模型，并研究其适合于计算机编程的计算方法。科学计算平台已经成为科学的研究必要的基础条件平台，有力地推动了科学的研究的发展和工程技术的进步。

机器学习应用需要科学计算的支持。大部分科学计算应用的领域都需要用到机器学习算法，科学计算平台与机器学习之间的关系就像鱼与水的关系。现代机器学习研究与应用早已经离不开科学计算平台的支撑，科学计算平台也因为机器学习的迅猛发展而进入了全新的百家争鸣时代。

2.1.1 常用的科学计算软件

目前常用的科学计算软件有以下几种：

1. MATLAB

MATLAB 是一种用于数值计算、可视化及编程的高级语言和交互式环境。使用 MATLAB，可以分析数据、开发算法、创建模型和应用程序，通过矩阵运算、绘制函数和数据、实现算法、创建用户界面、连接其他编程语言等方式完成计算，比电子表格或传统编程语言（如 C/C++ 或 Java）更方便快捷。MATLAB 具有强大的数值计算功能，可完成矩阵分析、线性代数、多元函数分析、数值微积分、方程求解、边值问题求解、数理统计等常见的数值计算，同时它也能进行符号计算。

2. GNU Octave

GNU Octave 与 MATLAB 相似，它是自由软件基金会开发的一个自由再发布软件，以 John W. Eaton 为首的一些志愿者共同开发了叫作 GNU Octave 的高级语言，这种语言与 MATLAB 兼容，主要用于数值计算，同时它还提供了一个方便的命令行方式，可以数值求解线性和非线性问题，以及做一些数值模拟。

3. Mathematica

Mathematica 系统是美国 Wolfram 研究公司开发的一个功能强大的计算机数学系统。它提供了范围广泛的数学计算功能，支持在各个领域工作的人们做科学的研究的过程中的各种计算。这个系统是一个集成化的计算机软件系统，它的主要功能包括符号演算、数值计算和图形三个方面，可以帮助人们解决各种领域里比较复杂的符号计算和数值计算的理论和实际问题。

4. Maple

1980 年 9 月，加拿大滑铁卢大学的符号计算研究小组研制出一种计算机代数系统，取名为 Maple，如今 Maple 已演变成为优秀的数学软件，它具有良好的使用环境、强有力的符号计算能力、高精度的数字计算、灵活的图形显示和高效的可编程功能。Maple 在符号计算

方面功能强大，符号计算式可以直接以数学的形式来输入和输出，直观方便。

5. SPSS

SPSS 预测分析是 IBM 公司的产品，它提供了统计分析、数据和文本挖掘、预测模型和决策优化等功能。IBM 宣称，使用 SPSS 可获得 5 大优势：商业智能，利用强大而简单的分析功能，控制数据爆炸，满足组织灵活部署商业智能的需求，提升用户期望值；绩效管理，指导管理战略，使其朝着最能盈利的方向发展，并提供及时准确的数据、场景建模、浅显易懂的报告等；预测分析，通过发现细微的模式关联，开发和部署预测模型，以优化决策制定；分析决策管理，一线业务员工可利用该系统，与每位客户进行互动，从中获取丰富信息，提高业务成绩；风险管理，在合理的前提下，利用智能的风险管理程序和技术，制定规避风险的决策。

6. R

R 语言是主要用于统计分析、绘图的语言和操作环境。R 目前由“R 开发核心团队”负责开发，它是基于 S 语言的一个 GNU 项目，语法来自 Scheme，所以也可以当作 S 语言的一种实现，虽然 R 主要用于统计分析或者开发统计相关的软件，但也可用作矩阵计算，其分析速度堪比 GNU Octave 甚至 MATLAB。R 主要是以命令行操作，网上也有几种图形用户界面可供下载。R 内建多种统计学及数字分析功能，还能透过安装套件（Packages）增强。

7. NumPy、SciPy、matplotlib 等 Python 科学计算平台

Python 是一种面向对象的、动态的程序设计语言，它具有非常简洁而清晰的语法，既可以用于快速开发程序脚本，也可以用于开发大规模的软件，特别适合于完成各种高层任务。随着 NumPy、SciPy、matplotlib 等众多程序库的开发，Python 越来越适合用于科学计算。NumPy 是一个基础科学的计算包，包括：一个强大的 N 维数组对象封装了 C++ 和 Fortran 代码的工具、线性代数、傅立叶转换和随机数生成函数等其他复杂功能的计算包。SciPy 是一个开源的数学、科学和工程计算包，能完成最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解等计算。matplotlib 是 Python 最著名的绘图库，它提供了一整套和 MATLAB 相似的命令 API，十分适合交互式制图，它也可以方便地用作绘图控件，嵌入 GUI 应用程序中。

2.1.2 本书使用的工程计算平台

MATLAB、Mathematica、Maple、SPSS 等软件功能齐全、界面友好，同时内含多种强大的软件包，但价格昂贵，它们都是商业化软件，GNU Octave、R 和 Python 科学计算包作为开源免费的工程计算平台，会是不错的选择。

本书选择 R 和 Python 科学计算包作为工程计算平台，Python 和 R 都能在多种操作系统平台上运行。Python 是一种非常流行的脚本语言，用户较多，容易掌握，也有成熟并行计算框架 dispy 等，测试成功的单机机器学习算法稍加修改就能应用于大规模分布式计算的工程之中。正是因为 Python 具有如此之多的优点，Google 内部也经常使用它。R 语言内置

大量统计分析包，能访问部分系统函数，核心为解释执行的语言，大部分用户可见的 R 函数由 R 语言本身编写，出于效率原理，计算密集型任务通过在运行时链接与调用 C、C++、FORTRAN 代码完成。此外，通过 Rcpp 能把丰富的 R 环境与 C/C++ 等结合，将 R 的 API 与数据对象封装成类以及类的方法，供外部 C++ 程序调用。

2.2 计算平台的配置

本章将以 Windows 平台和 Linux 平台为例，讲解 R 和 Python 科学计算平台的配置。Python 和 R 具有跨平台运行的特点，Windows 平台编写的 Python 和 R 代码只需修正兼容性问题即可正常运行在类 UNIX 平台上，如：中文字符的 UTF8 与 GBK 转换、Windows 系统与类 UNIX 平台的文件路径差异等。

2.2.1 Numpy 等 Python 科学计算包的安装与配置

Python 科学计算包有两种安装方式，即：分别安装科学计算平台内的软件包和安装 WinPython 集成计算包。

1. 分别安装科学计算平台内的软件包

先安装 Python，关于它的版本，推荐使用 2.7 版本，然后安装 NumPy、SciPy、matplotlib 等 Python 软件包，它们都有 Windows 系统下的安装包。

Python 安装包的下载页面为 <http://www.python.org/download/>，选择 2.7 版本的 Windows 安装可执行文件下载即可。

NumPy 安装包下载页面为 <https://pypi.python.org/pypi/numpy>，下载 Windows 版本的安装可执行文件即可。

SciPy 安装包下载页面为 <https://pypi.python.org/pypi/scipy/>，该软件包目前没有 Windows 版本的安装执行文件，要用传统的 Python 安装第三方软件包的方式安装，将安装包下载解压，然后在命令行进入解压目录，输入以下命令：

```
python setup.py install
```

Matplotlib 软件包的下载页面为 <http://matplotlib.org/downloads.html>，下载 Windows 版本的安装可执行文件即可，注意应下载 Latest stable version 对应的软件包。Windows 版本的安装可执行文件通常命名格式为：产品名称 + 平台名称 +CPU 型号 + 版本号。以 Matplotlib 为例，打开其下载页面，如图 2-1 所示。

假设计算机的 CPU 是 32 位，Python 版本号为 2.7，则下载安装 matplotlib-1.3.0.win32-py2.7.exe，如

Downloads

1.3.0 — Latest stable version

- [matplotlib-1.3.0.tar.gz](#)
- [matplotlib-1.3.0.win-amd64-py2.6.exe](#)
- [matplotlib-1.3.0.win-amd64-py2.7.exe](#)
- [matplotlib-1.3.0.win-amd64-py3.2.exe](#)
- [matplotlib-1.3.0.win-amd64-py3.3.exe](#)
- [matplotlib-1.3.0.win32-py2.6.exe](#)
- [matplotlib-1.3.0.win32-py2.7.exe](#)
- [matplotlib-1.3.0.win32-py3.2.exe](#)
- [matplotlib-1.3.0.win32-py3.3.exe](#)

图 2-1 Matplotlib 下载页面

果 CPU 是 64 位的，Python 版本号为 2.7，则下载安装 `matplotlib-1.3.0.win-amd64-py2.7.exe`。

在类 UNIX 平台上（以 UBUNTU 为例），可使用下面的命令安装 Python 及相关科学计算包：

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython iipython-notebook python-pandas python-sympy python-nose
```

2. 安装 WinPython 集成计算包

WinPython 集成计算包集成了 Numpy 等第三方 Python 科学计算库，安装 WinPython 后，Numpy 等计算库和 Python 2.7 会一同被安装。此外，WinPython 附带一款非常不错的 IDE 开发调试环境：Spyder，如图 2-2 所示是 Spyder 的界面截图。

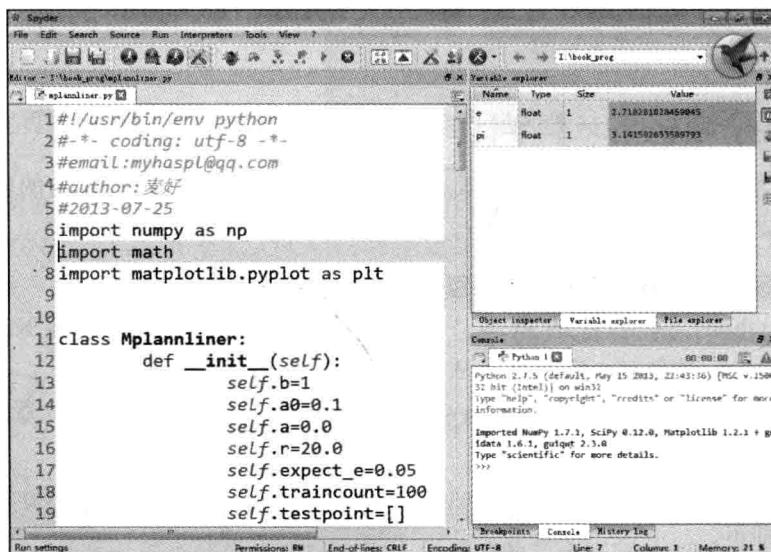


图 2-2 Spyder 界面

在图 2-2 所示的界面中，右上角是类似于 MATLAB 的“工作空间”，可很方便地观察和修改变量（包含多维数组）的值，同时还拥有方便用户的智能代码（Call-Tips 和 Auto-Complete）功能，如图 2-3 所示。

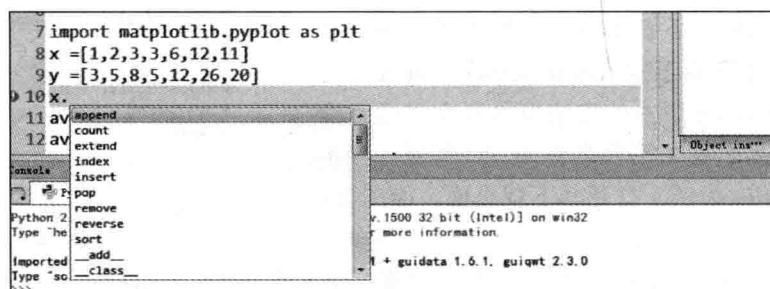


图 2-3 智能代码功能

在 IDE 开发窗口下方的 Console 栏可以使用 pdb (类似于 C 语言的 GDB 调试工具) 调试 Python 代码, 也可以通过 Spyder 的调试菜单进行调试。下面是 pdb 调试工具的使用帮助:

```
>>> debugfile(r'K:\book_prog\zxecf.py', wdir=r'K:\book_prog')
> k:\book_prog\zxecf.py(7)<module>()
-> import matplotlib.pyplot as plt
(pdb) help
Documented commands (type help <topic>):
=====
EOF      bt        cont     enable   jump    pp       run      unt
a         c        continue exit     l       q       s       until
alias    cl        d        h       list    quit    step    up
args     clear    debug    help    n       r       tbreak  w
b        commands disable ignore next   restart u       whatis
break   condition down    j       p       return unalias where
```

常用的 pdb 调试命令如下:

- h(elp): 打印当前版本 pdb 可用的命令。
- disable/enable: 禁用 / 启用断点。
- n(ext): 让程序运行下一行。
- c(ont(inue)): 让程序正常运行, 直到遇到断点。
- j(ump): 让程序跳转到指定的行数。
- b(reak): 设置断点, 例如 “b 23”, 就是在当前脚本的 23 行打上断点, 函数名也可作为参数。
- condition: 设置条件断点。下面语句就是对第 5 个断点加上条件 $x \geq 8$:

```
(Pdb) condition 5 x>=8
```

- cl(ear): 清除指定参数的断点或所有断点。
- p: 打印某个变量。比如:

```
(Pdb) p _file
u' ./pic/dog.jpg'
```

- ! : 感叹号后面跟着语句, 可以直接改变某个变量。
- q(uit): 退出调试。

综上所述, 在 Spyder 的帮助下, 能更高效地开发与调试 Python 代码, 因此笔者推荐在开发环境中安装 WinPython, 方便快捷, 有利于机器学习算法代码的编写。

2.2.2 OpenCV 安装与配置

OpenCV 是 Intel 开源计算机视觉库, 它由一系列 C 函数和少量 C++ 类构成, 实现了图像处理和计算机视觉方面的很多通用算法。OpenCV 拥有包括 300 多个 C 函数的跨平台的中高层 API。它不依赖于其他的外部库 (尽管也可以使用某些外部库), 对工程应用来说, OpenCV 是一个非常好的计算平台, 因为它遵守 BSD 开源协议, 对非商业应用和商业应用都是免费。

OpenCV 的主要功能有：图像数据操作，图像 / 视频的输入输出，矩阵 / 向量数据操作及线性代数运算，支持多种动态数据结构、基本图像处理、结构分析、摄像头定标、运动分析、目标识别、基本的 GUI 和图像标注。而且，OpenCV 提供了官方的 Python 接口，其使用方法和 C 语言接口基本一致，只是一些函数和结构体可能会有不同。另外，函数通过参数来返回值时一次会返回多个值。

在 Windows 上下载安装 OpenCV 的可执行文件后可直接运行，下载页面为 <http://opencv.org/downloads.html>。

其在 Linux 平台上的安装方式在 OpenCV 官网上有介绍，具体安装顺序如下：

(1) 安装基本软件包。GCC 4.4.x 或更高版本、CMake 或更高版本、Git、GTK+2.x 或更高版本、including headers (libgtk2.0-dev)、pkgconfig、Python 2.6 或更高版本、NumPy 1.5 或更高版本、python-dev、python-numpy、ffmpeg 或 libav 开发包、libavcodec-dev、libavformat-dev、libswscale-dev。

(2) 安装可选软件包。libdc1394 2.x、libjpeg-dev、libpng-dev、libtiff-dev、libjasper-dev。

(3) 在 <http://opencv.org/downloads.html> 下载其源代码，解压后，进入目录以源代码编译方式安装 OpenCV。

```
$cd ~/opencv
$mkdir release
$cd release
$cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
$make
$sudo make install
```

2.2.3 mlpy 安装与配置

mlpy 是基于 NumPy/SciPy 和 GSL 构建的 Python 模块，它提供了高层函数和类，允许使用少量代码来完成复杂的分类、特征提取、回归、聚类等任务。mlpy 为免费软件，建立在 GPL3 开源协议之上。

mlpy 在 Windows 下的安装方式较简单，可以直接在下面网址下载可执行文件安装：

<http://sourceforge.net/projects/mlpy/files/>

在类 Linux 平台上，其安装方法稍稍复杂一些，以 Linux、OSX 和 FreeBSD 为例，安装配置 mlpy，需要先安装配置好以下软件：

- GCC
- Python 且版本 ≥ 2.6 或为 3.X
- NumPy 且版本 $\geq 1.3.0$
- SciPy 且版本 $\geq 0.7.0$
- GSL 且版本 ≥ 1.11

然后，在上面网址中找到 mlpy 源代码包下载，并解压安装。假设 GSL 头文件和库文件没有安装在系统的标准位置，在这种情况下，mlpy 的安装方式如下：

```
$python setup.py build_ext --include-dirs=/path/to/header --rpath=/path/to/lib
$python setup.py install
```

如果 GSL 安装在标准位置，则只需要运行上述命令中的最后一行。

OpenCV 官方提供了 Python 绑定库，以 Python2.7 为例讲述了安装绑定库的方法，在 Windows 下，将它复制到 Python 的目录下，将 opencv\build\python\2.7 下的 cv2.pyd 文件复制到 python-2.7.5\Lib\site-packages 目录下即可。在 Linux 下安装了 Python 后，要确保 usr/lib/python2.7/site-packages 下有 cv.py 和 cv2.so 文件，如果没有，将这两个文件复制过来即可。

2.2.4 BeautifulSoup 安装与配置

BeautifulSoup 是用 Python 写的一个 HTML/XML 的解析器，它可以很好地处理不规范标记，并生成剖析树，通常用来分析“爬虫”抓取的 Web 文档，或者直接充当部分“爬虫”的角色。对于不规则的 HTML 文档，有补全功能。有了它，解析与分类网页就方便多了，节省了开发者的很多时间和精力。

安装 BeautifulSoup 很简单，在 Windows 平台和 Linux 平台上都是使用传统第三方库安装方式。首先下载 BeautifulSoup 源码，其官网为：<http://www.crummy.com/software/BeautifulSoup/>。

然后解压后运行以下命令：

```
python setup.py install
```

此外，在 UBUNTU 下还可以使用系统包管理器安装。

```
$ apt-get install python-bs4
```

2.2.5 Neurolab 安装与配置

NeuroLab 是一个简单而强大的、用 Python 编写的神经网络库，包括基础神经网络、训练算法，并具有弹性的构架，可创建其他网络，它用纯 Python 和 numpy 写成。API 的使用与 MATLAB 的神经网络工具箱类似，具有弹性的网络配置和学习算法，可以改变神经网络和学习算法的类型、训练、误差、初始函数和激活函数等神经网络参数。

Windows 和 Linux 下的安装方式如下。

首先在下面的页面下载 Neurolab：

<http://code.google.com/p/neurolab/downloads/list>

然后解压后运行如下命令：

```
python setup.py install
```

2.2.6 R 安装与配置

R 的原始码可自由下载使用，也有已编译的执行档版本可以下载，可在多种平台下运行，包括类 UNIX（包含 FreeBSD 和 Linux）、Windows 和 MacOS。

WINDOWS 安装方式如下。

首先访问其官网下载页面：

<http://ftp.ctex.org/mirrors/CRAN/>

然后下载安装可执行文件安装即可。

UBUNTU 下的安装方式如下：

```
$ sudo apt-get update
$ sudo apt-get install r-base
$ sudo apt-get install r-base-dev
```

2.3 小结

“不要重复造轮子” (Stop Trying to Reinvent the Wheel)，这可能是每个软件工程师入行时被告知的第一条准则。在轮子适合“机器学习”这台车的情况下，机器学习算法才能跑得更好，适合的科学计算平台就是机器学习的“轮子”。

笔者认为，作为机器学习这驾马车的“轮子”应该具备以下特征：

- 开源免费，且开源协议友好，例如：LGPL 协议或 BSD 协议，这样更有利于商业应用。
- 平台有文档，接口规范，能实际代码用例最好。
- 配置简单灵活，支持的操作系统平台多，运行速度快。
- 代码结构清晰、简单，便于使用者修改这个“轮子”，通俗地说：移植性强。

本书采用的计算平台在本章都一一列出其安装和配置方法。算法是一种计算思维的描述，万变不离其宗，好的工具原理都差不多。

也许随着时间的推移，算法在改进，更好的“轮子”将出现，所以不一定采用本书上所写的这些平台作为机器学习实验和应用的工具，但有一条原则：功能强大的计算平台不一定适合所有的工程，一切以适用为准。

第二部分

基础篇

合抱之木，生于毫末；九层之台，
起于累土；千里之行，始于足下。

——老子

第3章

机器学习数学基础

美国麻省理工学院的约翰·麦卡锡在 1955 年的达特矛斯会议上提出：人工智能就是要让机器的行为看起来就像人所表现出的智能行为一样。现代有一种观点，把人工智能分为了弱人工智能和强人工智能。维基百科是这样解释这两种智能的。

强人工智能：强人工智能观点认为有可能制造出真正能推理（Reasoning）和解决问题（Problem solving）的智能机器，并且，这样的机器将被认为是有知觉的，有自我意识的。强人工智能可以有两类，类人的人工智能，即机器的思考和推理就像人的思维一样；非类人的人工智能，即机器产生了和人完全不一样的知觉和意识，使用和人完全不一样的推理方式。

弱人工智能：弱人工智能观点认为不可能制造出真正能推理和解决问题的智能机器，这些机器只不过看起来像是智能的，但是并非真正拥有智能，也不会有自主意识。

目前人类主要的精力放在了弱人工智能研究方面，而弱人工智能则主要依托数学理论来解决机器学习的问题。其实，部分所谓的强人工智能也建立在数学分析的基础上。因此，大凡说到机器学习，总能看到一堆的数学公式来解说其算法，但讲解数据结构的书很少能看到数学公式，如果说数据结构描述了软件设计思维，那么机器学习就是描述了数学思维。

3.1 数学对我们有用吗

机器学习算法具有坚实的数学理论支持。机器学习的应用建立在科学计算的基础上，而数学计算则是科学计算的主要组成部分。在机器学习研究的各个领域（包括模式识别、数据分析、自动制造、专家系统等）都要应用到数学，数学是机器学习算法的基础。

数学是一切哲学、科学的基础，数学与软件是永远分不开的话题。我们应清醒地认识到，大数据时代已经到来，商业智能等机器学习技术开始普及，这些技术从大学的研究室和讲台走入了社会，应用到实际工程中了。因此，从某种意义上说：数学架起了从软件设计到智能计算的桥梁。

下面看看 Common Lisp 专家 Peter Seibel 对 Google 公司首席 Java 架构师 Joshua Bloch

的部分访谈 (Peter Seibel 写, 郝培强翻译), 里面谈到了数学与软件之间的关系, 笔者认为这是目前网上流传的最权威的关于数学和软件关系的定义:

Seibel: 你认识有哪位伟大的程序员不会数学或者没有接受过良好的数学教育吗? 要成为一个程序员, 学习微积分、离散数学和其他的数学知识真的那么重要吗? 还是做程序员只需要一种思想方式, 即使没有受过这些数字训练, 也能拥有?

Bloch: 我觉得是思想方式, 学不学数学都能拥有这种思想。但是学一下确实有好处。我曾有个同事叫 Madbot Mike McCloskey。他很懂数学, 但是没有学过数论。他重写了 BigInteger 的实现。原来的实现是 C 语言函数包的封装, 他发誓用 Java 重写, 要达到基于 C 语言版本的速度。后来他做到了。为此他学了大量的数论知识。如果他的数学不行, 他肯定搞不定这个项目, 而如果他本来就精通数论, 就无需费力去学习了。

Seibel: 但是, 这本来就是个数学问题啊。

Bloch: 对, 这个例子不恰当。但是, 我相信即使是跟数学无关的问题, 学习数学培养出的思维方式对编程来说也是必不可少的。例如, 归纳证明法和递归编程的关系非常紧密, 你不理解其中一个, 就不可能真正理解另外一个。你可能不知道术语的基本情况和归纳假设, 但是如果你不能理解这些概念, 你就没有办法写出正确的递归程序。所以, 即使是在与数学无关的领域内, 不理解这些数学概念的程序员也会遇到很多困难。

你刚才提到了微积分, 我觉得它不那么重要。可笑的是这么多年来似乎已经成为了一种思维定势了, 只要你受过大学教育, 那么人们就认为你应该懂微积分。因为微积分中有很多美妙的思想, 可以让人展开无穷的想象。

但是, 你可以以连续或者离散这两种不同的方式思维。我觉得对程序员来说, 精通离散思维更为重要。例如我刚提到的归纳证明法。你可以证明一种假设对所有整数都成立。证明过程就像施魔法一样。首先证明它对一个整数成立, 然后证明针对这个整数成立意味着针对下一个整数也成立, 这样就能证明它适用于全部整数。我认为对程序员来说这比理解极限的概念要重要得多。

好在我们无需选择。大学课程里这两样都教得不少。所以即使你用微积分用得没离散数学那么多, 学校里还是会教授微积分的。但是我认为离散的东西比连续的东西更重要。

综上所述, 程序员是以程序设计语言为工具, 编程解决特定问题的, 而编程的基础是计算机科学, 计算机科学的基础是数学。数学对程序员的作用具体表现在以下方面:

1. 培养编程思想

目前流行的编程语言呈现高度同质化的趋势, 只要学会某种语言, 其他语言只是语法和接口的变化, 但软件设计的编程思想却不尽相同。编写计算机程序的目的是为了解决实际问题, 严谨的思维模式是高效解决问题的关键。数学天生具有严谨性, 其基本要素是: 逻辑和直观、分析和推理、共性和个性, 这些也是编程思想的精髓。程序对问题的解决过程蕴含着数学思想方法。

美国著名数学教育家波利亚说过, 用数学思维解决问题就意味着要善于解题, 当我们解题时遇到一个新问题, 总想用熟悉的题型去套, 这只是满足于解出来, 只有对数学思想、数

学方法理解透彻及融会贯通时，才能提出新看法和巧解法。因此，我们在软件编码时要有意识地应用数学思想去分析问题、解决问题，培养数学头脑。

2. 提高算法效率

数学与软件算法相辅相成，数学是软件算法的灵魂，软件算法是数学的工具。数学的证明与求解过程往往是推导计算的过程，很多推导计算过于复杂，必须用程序算法验证，数学推导和算法设计有着密切关系；计算机算法的过程需要数学语言进行描述，算法效果的评估也需要数学方法，特别复杂的算法效率评测甚至需要建立专门的数学模型。

《纽约时报》2011年报道，著名的摩尔定律归纳了硬件技术进步的速度，而软件开发的突飞猛进推翻了这一定律。德国科学家和数学家马丁·格罗斯彻对15年之久的重要生产任务进展进行了调查，结果表明，在这15年里，运算完成速度提高了4300万倍，其中1000倍来自于处理器速度提高，43000倍则来自软件算法效率的改进。

唐纳德·克努特（经典著作《计算机程序设计艺术》的作者，此书被认为与数学著作的《几何学原理》相当）是算法和程序设计技术的先驱者。他大学一年级的暑假接触到IBM650，钻研使用手册后，他对数学产生了深厚的兴趣。一年后，他改学数学，从此与计算机结缘。他的第一个计算机程序也与数学相关：为他所在的校篮球队设计公式，根据球员在每场比赛中的得分、助攻、抢断、篮板球、盖帽等多项统计数字，对球员进行综合评估，并编写程序实现这个公式。

3.2 机器学习需要哪些数学知识

掌握机器学习算法至少需要以下几种数学的基本知识。

1. 微积分

微积分的诞生是继欧几里得几何体系建立之后的一项重要理论，它的产生和发展被誉为“近代技术文明产生的关键之一，它引入了若干极其成功的、对以后许多数学的发展起决定性作用的思想”。微积分学在科学、经济学和工程学领域有广泛的应用，解决了仅依靠代数学不能有效解决的问题。微积分学建立在代数学、三角学和解析几何学的基础上，包括微分学、积分学两大分支，包括连续、极限、多元函数的微积分、高斯定理等内容。

微积分在天文学、力学、化学、生物学、工程学、经济学、计算机科学等领域有着越来越广泛的应用，比如：在医疗领域，微积分能计算血管最优支角，将血流最大化；在经济学中，微积分可以通过计算边际成本和边际利润来确定最大收益；微积分可用于寻找方程的近似值；通过微积分解微分方程，计算相关的应用，比如，宇宙飞船利用欧拉方法来求得零重力环境下的近似曲线等。

在机器学习和数据分析领域，微积分是很多算法的理论基础，如：多层感知器神经网络算法。多层感知器是一种前馈人工神经网络模型，算法分为两个阶段：正向传播信号、反向传播误差。

正向传播信号阶段是对样本的学习阶段，输入的信息从输入层传入，经各个隐层计算后

传至输出层，计算每个单元的实际值，向各层各单元分摊产生的误差；反向传播误差阶段通过网络输出与目标输出的误差对网络进行修改审查，将正向输出的误差再传播回各层进行权重调整，直到误差最小化或达到规定的计算次数。微积分理论在多层感知器模型中运用较多，下面是3个应用的例子。

(1) 非线性激活函数是中间隐藏层的精髓，由于这些非线性函数的帮助，神经网络才能对线性和非线性模型进行学习，训练成功的网络能对待解决问题进行拟合和仿真。非线性激活函数要求处处可微，主要有Logistic函数和双曲正切函数。

Logistic函数定义为：

$$\phi(t) = \frac{1}{1+\exp(-a \cdot t)}$$

双曲正切函数定义为：

$$\varphi(t) = \tanh(a \cdot t) \frac{e^{a \cdot t} - e^{-a \cdot t}}{e^{a \cdot t} + e^{-a \cdot t}}$$

(2) 权值更新规则。权值更新规则定义为：

$$w(n+1) = w(n) + \eta \cdot \Delta w$$

其中 ΔW 基于最速下降法，定义为：

$$\Delta w = -\frac{\partial \varepsilon(w)}{\partial w} = -e(n) \frac{\partial \varepsilon(n)}{\partial w}$$

(3) 神经元局部梯度。梯度是一个向量场，标量场中某一点上的梯度指向标量场增长最快的方向，梯度的长度是这个最大的变化率。神经元局部梯度定义为：

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)}$$

2. 线性代数

线性代数是高等数学中的一门成熟的基础学科，它内容广泛，不但包含行列式、矩阵、线性方程组等初等部分，而且包括线性空间、欧式空间、酉空间、线性变换和线性函数、 λ -矩阵、矩阵特征值等更深入的理论，线性代数在数学、物理学、社会科学、工程学等领域也有广泛的应用。

线性代数理论是计算技术的基础，在机器学习、数据分析、数学建模领域有着重要的地位，这些领域往往需要应用线性方程组、矩阵、行列式等理论，并通过计算机完成计算。下面是几个应用线性代数的例子。

(1) 人口模型描述人口系统中人的出生、死亡和迁移随时间变化的情况，以及它们之间定量关系的数学方程式或方程组，分为连续模型和离散模型。其中离散模型适合于计算机仿真。在人口离散模型中，用 $x_0(t), x_1(t), x_2(t), \dots, x_m(t)$ 表示 t 时刻的年龄构成，其中 $x_i(t)$ 表示 t 年代年满 i 周岁但不到 $i+1$ 周岁的人口数，写成向量形式如下：

$$x(t) = \begin{Bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_m(t) \end{Bmatrix}$$

则离散人口模型可写成：

$$\left. \begin{aligned} x(t+1) &= H(t)x(t) + \beta(t)B(t)x(t) + g(t) \\ x(t_0) &= x_0 \end{aligned} \right\}$$

式中 $H(t)$, $B(t)$ 为以下相应维数的矩阵：

$$H(t)=\begin{pmatrix} 0 & & & \\ 1-\mu_1(t) & 0 & 0 & \\ & 1-\mu_2(t) & 0 & \\ 0 & \ddots & \ddots & \\ & & 1-\mu_{m-1}(t) & 0 \end{pmatrix}$$

$$B(t)=\begin{pmatrix} 0 \cdots b_{a1}(t) & b_{a1+1}(t) \cdots b_{a2}(t) & 0 \cdots 0 \\ & \ddots & \\ & & 0 \end{pmatrix}$$

式中 $\mu_i(t)(i=0, 1, \dots, m-1)$ 称为按龄死亡率, m 为人类能活到的最高年龄。

在这个模型中, 通过矩阵的形式, 将时间、出生、死亡和迁移 4 个因素及它们之间的定量关系进行完全描述。

(2) 投入产出技术是研究一个经济系统各部门间的“投入”与“产出”关系的数学模型, 该方法最早由美国著名的经济学家瓦·列昂捷夫提出, 是目前比较成熟的经济分析方法。投入产出数学模型根据投入产出原理建立的经济数学模型, 揭示国民经济各部门、再生产各环节之间的内在联系, 进行经济分析、预测和安排预算计划。

投入产出分析通常从投入产出表分析开始。投入产出表以数学方程式的形式来反映客观经济运行过程和经济结构, 它是根据投入产出表所反映的经济内容, 利用线性关系而建立起来的两组线性方程组。其中, 行模型根据投入产出表的横行关系建立经济数学模型, 其经济含义是揭示国民经济各部门生产的货物和服务的使用去向, 研究产出分配问题; 列模型根据投入产出表的纵列建立经济数学模型, 其经济含义是揭示国民经济各部门生产经营过程中发生的各种投入, 研究国民经济各部门生产货物和服务的价值形成问题。

(3) 自回归模型是统计上一种处理时间序列的方法, 从回归分析中的线性回归发展而来, 用同一变量例如 x 的前期进行预测(即 x_1 至 x_{t-1} 预测本期 x_t 的表现), 并假设它们为线性关系, 模型中 X 的当期值等于若干个后期值的线性组合, 加常数项, 加随机误差, 其公式定义为:

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t$$

其中: c 是常数项; ε_t 被假设为平均数等于 0、标准差等于 σ 的随机误差值; σ 被假设为对于任何的 t 都不变。

(4) 支持向量机 SVM (Support Vector Machine) 是一种小样本的机器学习方法, 它通过非线性映射, 把样本空间映射到一个高维乃至无穷维的 Hilbert 特征空间中, 使得非线性可分转化为在特征空间中的线性可分。SVM 方法的理论证明、核函数设计等都需要线性代

数理论的支持。

SVM 定义了以下两个超平面：

$$w \cdot x - b = 1$$

$$w \cdot x - b = -1$$

试图使它们之间没有任何样本点，且这两个超平面之间的距离最大，这样分类问题转变为二次规划最优化问题，在约束条件下最小化 $|w|$ ；然后用标准二次规划技术标准和程序解决，最终表示为以下训练向量的线性组合：

$$w = \sum_{i=1}^n \alpha_i c_i x_i$$

式中大于 0 的 α_i 对应的 x_i 就是支持向量。

3. 概率论

概率论是研究随机性或不确定性现象的数学，用来模拟实验在同一环境下会产生不同结果的情况。下面这些概率理论是概率论的基础。

(1) 古典概率。拉普拉斯试验中，事件 A 在事件空间 S 中的概率 $P(A)$ 为：

$$P(A) = \frac{\text{构成事件 } A \text{ 的元素数目}}{\text{构成事件空间 } S \text{ 的所有元素数目}}$$

■ 条件概率

一事件 A 在一事件 B 确定发生后会发生的概率称为 B 给之 A 的条件概率，定义为：

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

■ 概率公理

公理 1: $0 \leq P(A) \leq 1 (A \in S)$

公理 2: $P(S) = 1$

公理 3: $P(A \cup B) = P(A) + P(B)$, 如果 $A \cap B = \emptyset$

(2) 概率分布包括二项分布、几何分布、伯努利分布、泊松分布、均匀分布、正态分布、指数分布等。样本空间随机变量的概率分布可用累积分布函数和概率密度函数进行分析。

随机变量 X 的累积分布函数定义为：

$F(x) = P(X \leq x)$ 其中， x 为任意实数

设 X 为连续型随机变量，其累积分布函数为 $F_X(x)$ ，若存在 $f_X(x)$ ，满足：

$$\forall -\infty < a < \infty, F_X(a) = \int_{-\infty}^a f_X(x) dx$$

则 $f_X(x)$ 是它的概率密度函数。

概率论在机器学习和数据分析领域有举足轻重的地位，比如马尔可夫链理论。马尔可夫链对于现实世界的很多现象都给出了解释，泊松过程是连续时间离散状态的马尔可夫链，布朗运动是连续时间连续状态的马尔可夫链等。

马尔可夫链在计算数学、金融经济、机器学习、数据分析等领域都有重要的应用，马尔

可夫链是数学中具有马尔可夫性质的离散时间随机过程，在给定当前知识或信息的情况下，仅使用当前的状态预测将来，在马尔可夫链的每一步，系统根据概率分布，从一个状态变到另一个状态或保持当前状态。

马尔可夫链是随机变量 X_1, X_2, X_3, \dots 的一个数列，这些变量所有可能取值的集合为状态空间， X_n 的值则是在时间 n 的状态。设 X_{n+1} 对于过去状态的条件概率分布可定义如下：

$$P(X_{n+1}=x|X_0, X_1, X_2, \dots, X_n)=P(X_{n+1}=x|X_n)$$

同理，可计算更多步的转移概率：

$$\begin{aligned} p(X_{n+2}|X_n) &= \int P(X_{n+2}, X_{n+1}|X_n) dX_{n+1} = \int P(X_{n+2}|X_{n+1}) P(X_{n+1}|X_n) dX_{n+1} \\ p(X_{n+3}|X_n) &= \int P(X_{n+3}, X_{n+2}|X_n) \int P(X_{n+2}|X_{n+1}) P(X_{n+1}|X_n) dX_{n+1} dX_{n+2} \end{aligned}$$

4. 统计学

统计学是收集、分析、表述和解释数据的科学，作为数据分析的一种有效工具，统计方法已广泛应用于社会科学和自然科学的各个领域。统计学与概率论联系紧密，前者以后者为理论基础。统计学主要分为描述统计学和推断统计学。描述统计学描绘或总结观察量的集中和离散情形，基础的数学描述包括了平均数和标准差等；推断统计学将资料中的数据模型化，计算它的机率并且做出对于母群体的推论，主要包括假设检定、对于数字特征量的估计、对于未来观察的预测、相关性预测、回归、变异数分析、时间序列、数据挖掘等。

无论是描述统计学还是推断统计学都是数据分析技术的基础。通过描述统计学方法，数据分析专家能对数据资料进行图像化处理，将资料摘要变为图表，分析数据分布特征。此外，还可以分析数据资料，以了解各变量内的观察值集中与分散的情况等。通过推断统计学方法，对数据未知特征做出以概率形式表述的推断，在随机抽样的基础上推论有关总体数量特征。

5. 离散数学

离散数学是数学的几个分支的总称，研究基于离散空间而不是连续的数学结构，其研究内容非常广泛，主要包括数理逻辑、集合论、信息论、数论、组合数学、图论、抽象代数、理论计算机科学、拓扑学、运筹学、博弈论、决策论等。

离散数学广泛应用于机器学习、算法设计、信息安全、数据分析等领域，比如：数理逻辑和集合论是专家系统的基础，专家系统是一类具有专门知识和经验的计算机智能程序系统，一般采用人工智能中的知识表示和知识推理技术，模拟通常由领域专家才能解决的复杂问题；信息论、数论、抽象代数用于信息安全领域；与信息论密切相关的编码理论可用来设计高效可靠的数据传输和数据储存方法；数论在密码学和密码分析中有广泛应用，现代密码学的 DES、RSA 等算法技术（包括因子分解、离散对数、素数测试等）依赖于数论、抽象代数理论基础；运筹学、博弈论、决策论为解决很多经济、金融和其他数据分析领域的问题提供了实用方法，这些问题包括资源合理分配、风险防控、决策评估、商品供求分析等。

以上是机器学习需要的核心数学知识，但不是全部知识。随着今后人类对机器学习的深入研究，将有更多的数学分支进入机器学习领域。因此，仅掌握大学数学知识是不够的，还需要向更高层次进军，对于非数学专业毕业的朋友来说，还应该学习其他数学分支理论，比

如说泛函分析、复变函数、偏微分方程、抽象代数、约束优化、模糊数学、数值计算等。

建议读者购买以下数学书籍，随时翻阅参考。

Finney, Weir, Giordano.《托马斯微积分》.叶其孝,王耀东,唐兢译.第10版.北京:高等教育出版社 2003-1

Steven J.Leon.《线性代数》.张文博,张丽静译.第8版.北京:机械工业出版社

William Mendenhall 等.《统计学》.梁冯珍,关静译.第5版.北京:机械工业出版社

Dimitri P. Bertsekas 等.《概率导论》.郑忠国,童行伟译.第2版.北京:人民邮电出版社

Kenneth H.Rosen 等.《离散数学及其应用》.袁崇义,屈婉玲,张桂芸译.第6版.北京:机械工业出版社

Eberhard Zeidler 等.《数学指南:实用数学手册》.李文林译.北京:科学出版社

它们都是机器学习所涉及的经典数学书,可以考虑将它们和《设计模式》、《算法导论》、《深入理解计算机系统》等经典算法书放在一起,作为案头必备书。

3.3 小结

机器学习以数学理论为基础,这里的数学理论主要是应用数学。应用数学是应用性较强的数学学科或分支的统称,数学本来起源于实际应用的需要,应用一直是数学的发展动力之一,一种数学理论和一门数学学科的生命力的强弱,在很大程度上依赖于它有无应用需求,机器学习就是数学的应用领域之一。

应用数学是应用目的明确的数学理论和方法的总称,是纯数学(纯数学研究数学本身,不以应用为目的,以其严格、抽象和美丽著称,主要研究空间形式的几何类、离散系统的代数类、连续现象的分析类)的相反,包括微分方程、向量分析、矩阵、拉普拉斯变换、傅里叶变换、复变分析、数值方法、概率论、数理统计、运筹学、控制理论、组合数学、信息论等许多数学分支,也包括从各种应用领域中提出的数学问题的研究。随着计算机技术的发展,现在计算数学也加入了应用数学的行列。

一位 MIT 的牛人在 BLOG 中曾提到,数学似乎总是不够的,为了解决和研究工程中的一些问题,不得不在工作后,重新回到图书馆捧起了数学教科书。他深深感到,从大学到工作,课堂上学的和自学的数学其实不算少了,可是在机器学习领域总是发现需要补充新的数学知识。看来,要精通机器学习知识,必须在数学领域学习、学习、再学习,这一切都是很艰苦的。要学好机器学习必须做好艰苦奋斗的准备,坚持对数学知识的追求。

本章对机器学习中需要掌握的相关数学知识提出了要求,同时也推荐了有关数学书籍。不要因为学习数学麻烦、难度大就不去接触它,数学才是工程师软实力的体现。古人云:“磨刀不误砍柴工”,这个“刀”就是数学知识。

第4章

计算平台应用实例

4.1 Python 计算平台简介及应用实例

目前，科学计算平台很多，在本书中，使用 Python 编写的机器学习算法用到的计算平台有：Numpy 等科学计算包、OpenCV 的 Python 绑定库、mlpy 机器学习库、BeautifulSoup 网页解析库、Neurolab 神经网络库等，所有平台均为开源免费软件。本章将讲解这些计算平台的操作，并解析一些基础实例应用。

4.1.1 Python 语言基础

1989 年圣诞节期间，吉多·范罗苏姆为了打发假日时间，决心开发一个新的脚本解释程序，作为 ABC 语言的一种继承，就这样，Python 在吉多手中诞生了。Python 的设计哲学是优雅、明确、简单。Python 提供了丰富的 API 和工具，使程序员能使用 C 语言、C++、Cython 编写扩充模块。Python 编译器本身也可以被集成到其他需要脚本语言的程序中。此外，很多人把 Python 作为一种“胶水语言”使用，用它将其他语言编写的程序进行集成和封装。

Python 包含了一组完善而且容易理解的标准库，能够轻松地完成很多常见的任务，且代码语法简洁、清晰，使用缩进定义语句块，具备很高的可读性。由于 Python 语言具有简洁、易读以及可扩展性，在国内外用 Python 做科学计算的研究机构、商业公司日益增多，比如：三大经典科学计算库 Numpy、SciPy 和 matplotlib 均扩展了 Python，通过 Python 可以轻松完成快速数组处理、数值运算和绘图任务。

1. Python 基本数据类型

Python 是解释运行的动态语言，解释器的提示符为“>>>”。Python 的基本数据类型包括数字型、字符串型和列表，此外还可以用类型表示函数、模块、类型本身、对象的方法、编译后的 Python 代码、运行时信息等。

(1) 数字型，可将 Python 作为一个计算器使用，在计算过程中可使用“+”(加)、“-”

(减)、“*”(乘)、“/”(除)、“(”、“)”以及“%”(取余)等操作符。此外，Python 用“#”表示其后的内容是注释。下面代码演示了其基本的计算功能和注释的使用。

```
>>> 2+2
4
>>> # 本行是注释
... 2+8
10
>>> (50-5*6)/2
10
>>> 17/-4
-5
>>> 9%2
1
```

Python 使用“=”进行赋值操作，赋值操作不会返回任何结果，也可以在同一行中连接赋值，赋值语句将从右到左依次完成赋值。下面的代码对变量进行复数和实数的赋值。

```
>>> c=5.2-6.5j###复数
>>> c
(5.2-6.5j)
>>> c.real###实部
5.2
>>> c.imag###虚部
-6.5
>>>x = y = z = 0
```

变量在使用前，要处于定义状态（即已经赋值），否则会出错。下面代码试图对未定义的变量 myx 进行操作结果出错了。

```
>>> myx # 访问未定义的变量
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'myx' is not defined
```

在 Python 计算中可使用浮点数。如果在计算过程中出现了浮点数，则整型会自动转换为浮点型计算；如果全是整型，则计算结果也为整型。在下面例子中，第一行代码两个操作数均为整型，返回的结果并不会精确为整型；而在第二行代码中，第一个操作数 7.0 为浮点型，返回的结果为浮点数。

```
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
```

复数运算使用 (real+imagj) 的形式，也可使用 complex(real, imag) 创建一个复数对象，其中 real 表示实部，imag 表示虚部。下面是一些复数计算的例子。

```
>>> 2+6J
(2+6j)
>>> 2-6J
(2-6j)
```

```
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

变量“`_`”表示刚计算的结果。下面代码通过计算 $(3+2) \times 6$ 演示了该变量的使用。

```
>>> 3+2
5
>>> _ * 6
30
```

(2) 字符串型。Python 的字符串通常用单引号和双引号包围的形式表示，通过 `print` 语句可将字符串输出到输出设备中（默认情况下输出到屏幕）。此外，Python 2.0 及以后的版本支持 Unicode 字符串，中文字符一般使用 Unicode 编码（国际组织制定的容纳世界所有文字和符号的字符编码方案，用 0~0xFFFF 映射字符，最多可以容纳 1114112 个字符），在前面加 `u` 表示后面是 Unicode 字符串。下面的代码演示了 Unicode 字符的定义与使用、字符串的输出等操作。

```
>>> 'spam eggs'
'spam eggs'
>>> '"Yes," he said.'
'"Yes," he said.'
>>> print u"你好"
你好
>>> print u"机器学习"
机器学习
```

Python 的字符串可视为列表（数组），使用 “[索引]” 的方式能对它进行切片操作（索引从 0 开始），使用 “`+`” 可对字符串进行连接。下面的代码演示了字符串的连接、切片等操作。

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
>>> word[-2:]      #最后2个字符
'pA'
>>> word[:-2]       #除去最后2个字符以外的字符
'Hel'
```

Python 的字符串可使用转义字符，方法是在特殊字符前加上 “`\`”。主要的转义字符有：

- ＼ 单引号
- ＼" 双引号
- ＼a 发出系统响铃声

\b 退格符
 \n 换行符
 \t 横向制表符
 \v 纵向制表符
 \r 回车符
 \f 换页符
 \\ \
 \o 八进制数代表的字符
 \x 十六进制数代表的字符

\000 终止符, \000 后的字符串全部忽略

此外, Python 的字符串相比其他程序语言多了一种描述方式, 就是用 3 个引号标示字符串, 其功能是将字符串内容原样输出, 如果字符串本身包括换行, 则输出换行, 如果包括特殊字符, 则字符串无需使用转义字符。下面的代码演示了中文字符串的输出、转义字符的使用、字符串切片、统计长度、三引号使用等操作。

```
>>> 'doesn\'t'
"doesn't"
>>> "\\"Yes, \" he said."
'"Yes," he said.'
>>> print u"你好,Python\n机器学习"
你好,Python
机器学习
>>> print u"""你好,Python
... 机器学习"""
你好,Python
机器学习
>>> mystr= u"你好,Python\n机器学习"
>>> print mystr
你好,Python
机器学习
>>> print mystr[:5]
你好,Py
>>> print mystr[3:5]
Py
>>> len(mystr)###len函数计算字符串长度
14
```

Python 字符串的三引号表示方式意义重大, 用它可以在 CGI (CGI 允许 Web 服务器执行外部程序, 并将它们的输出发送给 Web 浏览器) 程序中轻松输出 HTML 代码。下面是 CGI 的 “Hello World” 程序:

```
#!/usr/bin/env python
print "Content-Type: text/html"
print
print """<html>
```

```
<body>
<h2>Hello World!</h2>
</body>
</html>
"""
```

(3) 列表。Python 可将不同类型(包括复合类型本身)的值组成复合类型,列表就是复合类型之一。与字符串相同,列表可以使用切片操作,其索引也是从 0 开始的。此外,统计列表的长度使用 len 函数,使用“*”将生成新列表,其元素由源列表重复填充,使用“+”可连接列表。下面的代码演示了列表的定义、连接、重复填充、切片、统计长度等操作。

```
>>> a = ['hello', 'world', 100, 1234]
>>> a
['hello', 'world', 100, 1234]
>>> a[:2] + [('-', 2*2)]##连接列表
['hello', 'world', '(', 4]
>>> mylist=[1,23,45]
>>> mylist
[1, 23, 45]
>>> mylist*2##重复填充列表
[1, 23, 45, 1, 23, 45]
>>> mylist[:2]##列表切片
[1, 23]
>>> x = [12, 13]
>>> y = [11, x, 14]
>>> len(y)##求列表长度
3
>>> y[1]
[12, 13]
>>> x[1][0]
12
```

列表和字符串也可称为序列,序列由若干个元素组成,每个元素的先后顺序明确,不能更改。

2. Python 语句

(1) 条件语句。if 语句的作用是判断条件是否成立,如果成立,则执行后面的语句块;if ... elif ... elif 语句可用于对多个条件进行判断,并执行最先满足条件的语句块;else 语句表示所有条件都不成立时执行。下面的例子展示了 if 语句的使用方法,功能是对输入数字的范围进行判断,并将其中的负数转变为 0。

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: -10
>>> if x < 0:
...     x = 0
...     print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
```

```

...     print 'Single'
... else:
...     print 'More'
...
Negative changed to zero
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3

```

(2) 循环语句。for 语句表示循环，它与 C 语言中的 for 语句略有不同。C 语言的 for 语句可定义步长和终止循环条件，而 Python 的 for 语句在 Python 的序列（列表、字符串等）中迭代，每次只操作其中一项。下面的代码在单词列表中迭代，每次迭代输出单词及其长度。

```

>>> # Measure some strings:
... words = ['cat', 'window', 'hello']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
hello 5

```

也可以在迭代过程中修改序列。下面的例子运行结果是在列表迭代过程中修改了列表本身。

```

>>> words = ['cat', 'window', 'hello']
>>> for w in words[:]:
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestratate', 'cat', 'window', 'defenestratate']

```

(3) range 函数。for 语句仅能在列表等序列中进行迭代，从表面上来看，它比 C 语言的 for 循环功能弱很多，其实不然，有了 range 函数，其功能远比 C 语言的 for 循环强大。range 函数可产生符合某种规律的列表等序列。下面代码产生 0~9 共 10 个数字，增长步长

为 1。

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`range` 函数还可以产生更复杂的序列，常用调用格式为：

```
range(起始值, 元素数量, 步长)
```

其中起始值和步长可以省略，起始值默认为 0，步长默认为 1。

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

下面的代码是 `range` 函数与 `for` 语句组合的应用，输出列表元素的索引及值。

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

下面的代码完成 1 至 10 中偶数的累加。

```
>>> mysum=0
>>> for i in range(1,10,2):
...     mysum=mysum+i
...
>>> mysum
25
```

(4) `break` 与 `continue`。`break` 语句结束本层循环，`continue` 语句忽略下面的语句继续本层的下次循环。下面的代码使用 `break` 语句，查找 2 至 10 以内的素数，如果不是素数，则分解因数。

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
```

```
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

下面的代码使用了 continue 语句，判断 50 至 60 之间的数哪些是奇数，哪些是偶数。

```
>>> for num in range(50, 60):
...     if num % 2 == 0:
...         print "偶数", num
...         continue
...     print "奇数", num
...
偶数 50
奇数 51
偶数 52
奇数 53
偶数 54
奇数 55
偶数 56
奇数 57
偶数 58
奇数 59
```

(5) while 循环。在 while 循环中会一直执行后面的语句块，直到条件不满足才终止。

下面的代码用于清理列表，并且仅保留奇数。

```
>>> a=range(20)
>>> for x in a[:]:
...     if x%2== 0: a.remove(x)
...
>>> a
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

(6) 函数定义。Python 使用 def 关键字定义函数。下面的代码定义了屏幕输出函数 show，参数是要输出的内容。

```
>>> def show(mess="hello"):
...     print mess
...
>>> show()
hello
>>> show("机器学习")
机器学习
```

下面的代码定义了斐波那契数列的计算函数 fib。

```
>>> def fib(n):
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
```

```

...
>>> fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

```

3. Python 的元组、集合以及字典

(1) tuple 元组。tuple 元组类似列表，不同的是它的内容不能修改。Python 元组的定义方式是使用括号包含元素或直接列举元素。下面的代码将定义 x 为元组类型，当对 x[0] (x 的第一个元素) 进行修改时，Python 解释器提示错误。

```

>>> x = 10, 20, 'learn'
>>> x[0]
10
>>> x
(10, 20, 'learn')
>>> x1,x2,x3=x
>>> x1
10
>>> x2
20
>>> x3
'learn'
>>> x[0]=90
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>

```

(2) Sets 集合。Python 使用方括号定义 Sets 集合，集合的特点是无重复元素，集合中的元素无先后顺序，也不能用索引进行管理。下面的代码定义了一个水果列表，通过 set 函数转换为集合，去除重复元素。

```

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])

```

下面的代码通过 in 操作检查成员与集合的关系。

```

>>> 'orange' in fruit ###集合类是否有成员 'orange'
True
>>> 'crabgrass' in fruit###集合类是否有成员 'crabgrass'
False

```

既然是集合，当然能对它进行集合的数学运算。Python 集合支持 union (联合或并)、intersection (交)、difference (差) 和 sysmmetric difference (对称差) 等操作，可通过“-”、“|”、“&”、“^”操作符计算集合的差集、并集、交集和对称差集。下面的代码演示了对集合 a 和 b 的运算。

```

>>> a = set('abracadabra')
>>> b = set('alacazam')

```

```

>>> a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                                #差集
set(['r', 'd', 'b'])
>>> a | b                                # 并集
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                                # 交集
set(['a', 'c'])
>>> a ^ b                                # 对称差集
set(['r', 'd', 'b', 'm', 'z', 'l'])

```

(3) Dictionaries 字典。Dictionaries (字典) 是一种散列结构，可理解为 Hash (散列) 类型。字典由若干个键值对组成，键值对是一种映射，一个键对应于一个值。键值对由两个部分组成，第一部分是键，键在字典中必须保持唯一，字典与列表和元组不同，列表和元组属于序列，元素以先后顺序来管理，而字典的元素是以键为单位对值进行管理的；第二部分为值，值与键一一对应，值可以彼此相同。

Python 使用花括号定义字典，使用类似索引的方式存取值，但索引为键。此外，可使用 del 操作删除键值对，使用 keys() 方法返回字典变量存储的所有键。下面的代码演示了一个学生学号的字典变量，以及如何对学生信息进行删除、查询等操作。

```

>>> tel = {'张三': 4098, '李四': 4139}
>>> tel
{'\xd5\xc5\xc8\xfd': 4098, '\xc0\xee\xcb\xc4': 4139}
>>> tel['张三']
4098
>>> del tel['张三']
>>> tel['张三']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: '\xd5\xc5\xc8\xfd'
>>> tel.keys()
['\xc0\xee\xcb\xc4']
>>> tel['王华']=1000
>>> tel.keys()
['\xcd\xf5\xbb\xaa', '\xc0\xee\xcb\xc4']
>>> print tel.keys()[0]
王华

```

上面代码中，“\bcd” 等为汉字在 Python 内部的编码。

4. Python 类

Python 语言有强大的面向对象编程能力。Python 和 C++ 等语言一样拥有类机制，Python 类的定义方式如下：

```

class 类名:
    #类成员变量
    变量A=A初始值
    变量B=B初始值
    .....

```

```

#下面定义了类成员函数
def __init__(self, 参数1, 参数2, ..., 参数n):
    #类构造函数
    .....
def __del__(self):
    #析构函数
    .....
def 方法1(self, 参数1, 参数2, ..., 参数n):
    #类的方法
    .....
    .....

```

下面的代码定义了一个复数类 Complex，同时定义了类的实例变量 x，最后输出该变量的实部和虚部。Complex 类很简单，在类构造函数中对实部成员和虚部成员进行赋值。

```

>>> class Complex:
...     r=0
...     i=0
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)

```

5. Python 异常处理

Python 的异常处理能力很强大，可准确反馈出错信息。在 Python 中，异常是对象，可对它进行操作。所有异常都是基类 Exception 的成员，从基类 Exception 继承，在 exceptions 模块中定义。Python 异常处理的格式如下：

```

try :
    #下面为可能发生异常的语句块
    .....
except 异常类型:
    #下面为处理异常的语句块
    .....

```

下面的代码将检查输入是否为有效数字。程序通过将输入转换成整型来测试是否为数字，如果不是数字，int 函数将触发异常 ValueError，异常处理程序提示“哦！输入不是有效数字，请重新输入（Oops! That was no valid number. Try again...）”，直到输入正确格式的数字后，程序才退出。

```

>>> while True:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops! That was no valid number. Try again..."
... 

```

前面演示了异常的被动触发，Python 还能主动触发异常，处理方式为：先通过 `raise` 语句抛出异常，然后用 `except` 捕捉异常。下面的代码演示 `raise` 主动抛出 `NameError` 异常后被捕捉，输出“An exception flew by!”后，继续抛出异常，以便给更外层的异常处理函数继续处理。

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print 'An exception flew by!'
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

以上简要介绍了 Python 编程语言的基本语法，它和 C++ 有几分相似，有一定编程基础的读者应该都能看懂。Python 是一门上手很快的编程语言，它与其他语言最大的不同就是它区分语句块时使用的不是括号，而是每行语句前面的空格数量。因此，Python 代码非常工整和漂亮，它严格遵守代码语句块的缩进原则，可依靠缩进判断语句块的范围。

4.1.2 Numpy 库

本书中介绍的机器学习算法大部分是调用 Numpy 库来完成基础数值计算的。下面了解一下 Numpy 库的基本使用方法。

1. ndarray 数组基础

Python 中用列表保存一组值，可将列表当成数组使用。此外，Python 有 `array` 模块，但它不支持多维数组，无论是列表还是 `array` 模块都没有科学运算函数，不适合做矩阵等科学计算。因此，Numpy 没有使用 Python 本身的数组机制，而是提供了 `ndarray` 数组对象，该对象不但能方便地存取数组，而且拥有丰富的数组计算函数，比如向量的加法、减法、乘法等。

使用 `ndarray` 数组，首先需要导入 Numpy 函数库，可以直接导入该函数库（本节频繁使用 Numpy 库的函数，因此采用这种方法）。

```
from numpy import *
```

或者指定导入库的别名。

```
import numpy as np
```

下面正式进入 Numpy 的数组世界。如果没有说明，所称数组均为 Numpy 的数组对象，与 Python 的列表和 `array` 模块无关。

(1) 创建数组。创建数组是进行数组计算的先决条件，可通过 `array()` 函数定义数组实例对象，其参数为 Python 的序列对象（比如列表）。如果想定义多维数组，则传递多层嵌套

的序列。例如下面这条语句定义了一个二维数组，其大小为(2, 3)，即共有2行，每行各3列。

```
a = np.array([[ 1., 7., 0.], [-2., 1., 2.]])
```

上面语句定义了如表4-1所示的数组。

表4-1 二维数组结构

1	7	0
-2	1	2

接着使用array()函数创建一个(2,3)大小的数组变量x。

```
>>> from numpy import *
>>> x=np.array([[ 1., 0., 0.], [ 0., 1., 2.]])
```

以刚才定义的x变量为例，来熟悉 ndarray 数组对象的主要属性。ndarray 数组对象拥有 ndarray.ndim、ndarray.shape、ndarray.size、ndarray.dtype、ndarray.itemsize、ndarray.data 等属性。

ndarray.ndim：数组的行数。

```
>>> x.ndim
2
```

ndarray.shape：数组的维数，返回的格式为(n,m)，其中n为行数，m为列数。

```
>>> x.shape
(2, 3)
```

ndarray.size：数组元素的总数。

```
>>> x.size
6
```

ndarray.dtype：数组元素的类型，比如：numpy.int32（32位整型）、numpy.int16（16位整型）及 numpy.float64（64位浮点型）。

```
>>> x.dtype
dtype('float64')
```

ndarray.itemsize：数组中每个元素占有的字节大小。

```
>>> x.itemsize
8
```

ndarray.data：数组元素的缓冲区。

```
>>> x.data
<read-write buffer for 0x0557EAE8, size 48, offset 0 at 0x0561BAE0>
```

下面是一个关于Numpy的ndarray数组的例子，演示了ndarray数组的基本操作。

首先，创建a和b两个数组对象。其中，a对象使用Numpy的arange函数产生了等差序列数组（Numpy的arange函数与Python的range函数类似，其参数依次为开始值、结束值、步长），并用reshape函数创建了指定形状的新数组。a的大小为(3,5)；b的大小为(1,3)。

```
>>> a = arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
```

接着读取 a 和 b 的主要属性，如：shape、ndim、dtype、itemsize 等。此外，下面的代码还演示了 type 函数的使用，type 函数会返回对象的类型，对于 ndarray 对象而言，其类型为 numpy.ndarray。

```
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
numpy.ndarray
>>> type(b)
numpy.ndarray
```

最后，对 a 和 b 对象重新赋值，以便进一步了解 ndarray 数组的元素类型。下面分别演示了 int32、float64 等类型的使用方法，最后演示了不常见的复数作为数组元素（数组 c 的元素）的情况。

```
>>> a = array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
>>> c = array([[1, 2], [3, 4]], dtype=complex)
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

(2) 特殊数组。Numpy 的特殊数组主要有以下几种：

- zeros 数组：全零数组，元素全为 0，使用 zeros 函数创建。
- ones 数组：全 1 数组，元素全为 1，使用 ones 函数创建。
- empty 数组：空数组，元素全近似为 0，使用 empty 函数创建。

下面的代码依次演示了全零数组、全1数组、空数组的创建方法。

```
>>> zeros( (3,4) )
array([[0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.]])
>>> ones( (2,3,4), dtype=int16 )
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],
       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
>>> empty( (5,3) )
array([[ 1.42185083e-299,   1.41906197e-299,   5.77420410e-300],
       [ 6.02082633e-300,   1.41971817e-299,   5.77379398e-300],
       [ 5.77440917e-300,   5.77386233e-300,   5.77440917e-300],
       [ 5.77386233e-300,   5.77440917e-300,   5.77386233e-300],
       [ 1.42130399e-299,   5.77440917e-300,   5.77406740e-300]])
```

(3) 序列数组。刚才已经提到过 `arange` 函数，它与 Python 的 `range` 函数相似，但它属于 Numpy 函数库，其参数依次为开始值、结束值、步长。此外，还可使用 `linspace` 函数创建等差序列数组，其参数分别为起始值、终止值、元素数量。下面的代码分别演示了 `arange` 函数和 `linspace` 函数的用法。

```
>>> arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> arange( 0, 2, 0.3 )
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
>>> linspace( 0, 2, 9 )          # 从0到2, 9个数字
array([ 0. ,  0.25,  0.5 ,  0.75,  1. ,  1.25,  1.5 ,  1.75,  2. ])
>>> x = linspace( 0, 2*pi, 100 ) # 从0到2*pi共100个数字
```

(4) 输出数组。可使用 `print` 输出 Numpy 的数组对象。下面的代码是一维数组的创建和输出。

```
>>> a = arange(6)                      # 一维数组
>>> print a
[0 1 2 3 4 5]
```

下面的代码是二维数组的创建和输出，从输出结果可清晰地看出 `b` 的大小为 (4,3)。

```
>>> b = arange(12).reshape(4,3)         # 二维数组
>>> print b
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

(5) 数组索引。Numpy 数组的每个元素、每行元素、每列元素都可以用索引访问，不过要注意索引是从 0 开始的。比如，某数组大小为 (2,3)，则第 2 行第 1 列元素的索引是 [1,0]。下面以三维数组为例，演示数组的创建、输出及索引。

首先创建三维数组 c，并输出其元素。

```
>>> c = arange(24).reshape(2,3,4)          # 三维数组
>>> print c
[[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]]
```

三维数组 c 可表示为如图 4-1 所示的形式。

[0,0,:]				[0,1,:]				[0,2,:]			
0	1	2	3	4	5	6	7	8	9	10	11
[1,0,:]				[1,1,:]				[1,2,:]			
12	13	14	15	16	17	18	19	20	21	22	23

图 4-1 三维数组 c

图 4-1 中单元格上方标注了 [0,0,:], [0,1,:], [0,2,:], [1,0,:], [1,1,:], [1,2,:], 索引中的“:”表示该维度内的所有元素。对照图 4-1，查找索引 [1,2,:] 处（第 2 行第 3 列）的所有元素和索引 [0,1,2] 处的元素，可得出元素为 [20,21,22,23] 和 6。下面编写代码验证一下。

```
>>> print c[1,2,:]
[20 21 22 23]
>>> print c[0,1,2]
6
```

(6) 数组运算。数组的加减乘除以及乘方运算方式为，相应位置的元素分别进行计算。比如：

数组加法：array([20,31,42,53])=array([20,30,40,50])+array([0,1,2,3])
 数组减法：array([20, 29, 38, 47])=array([20,30,40,50])-array([0,1,2,3])
 数组乘法：array([[2, 0],[0, 4]])=array([[1,1],[0,1]])*array([[2,0],[3,4]])
 数组乘方：array([0, 1, 2, 3]) 的二次方 =array([0, 1, 4, 9])
 数组除法：array([20., 15., 13.33333333, 12.5])=array([20,30,40,50])/array([1, 2, 3, 4])

下面的代码演示了数组的加、减、乘、除及更多运算（关键代码处注释了运算类型）。

```
>>> a = array([20,30,40,50])
>>> aa = arange(1, 5)
>>> a/aa##除法
array([ 20.          ,  15.          ,  13.33333333,  12.5        ])
>>>
>>> b = arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b##减法
>>> c
```

```

array([20, 29, 38, 47])
>>> b**2##乘方
array([0, 1, 4, 9])
>>> 10*sin(a)##数乘,sin函数为正弦函数
array([-9.12945251, -9.88031624, -7.4511316, -2.62374854])
>>> a<35##指定条件判断后生成相应的布尔数组
array([True, True, False, False], dtype=bool)
>>> A = array([[1,1],[0,1]])
>>> B = array([[2,0],[3,4]])
>>> A*B##乘法
array([[2, 0],
       [0, 4]])
>>> #dot表示乘积。对一维数组计算的是点积，对二维数组计算的是矩阵乘积
...#此处表示矩阵乘积
... dot(A,B)
array([[5, 4],
       [3, 4]])
>>> a = ones((2,3), dtype=int)##ones函数创建全1数组，指定元素类型为int
>>> b = random.random((2,3))##创建随机数组，指定大小为(2,3)
>>> a *= 3##数乘
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a##加法
>>> b
array([[3.69092703, 3.8324276, 3.0114541],
       [3.18679111, 3.3039349, 3.37600289]])
>>> a += b ##加法
>>> a
array([[6, 6, 6],
       [6, 6, 6]])
>>> a = random.random((2,3))
>>> a
array([[0.6903007, 0.39168346, 0.16524769],
       [0.48819875, 0.77188505, 0.94792155]])
>>> a.sum()##求和
3.4552372100521485
>>> a.min()##求最小值
0.16524768654743593
>>> a.max()##求最大值
0.9479215542670073

```

(7) 数组的拷贝。数组的拷贝分为浅拷贝和深拷贝两种，浅拷贝通过数组变量的赋值完成，深拷贝使用数组对象的 copy 方法。

浅拷贝只拷贝数组的引用，如果对拷贝进行修改，源数组也将修改。下面的代码演示了浅拷贝的方法。

```

>>> a=ones((2,3))
>>> a
array([[1., 1., 1.],

```

```
[ 1.,  1.,  1.])
>>> b=a###b为a的浅拷贝
>>> b[1,2]=2
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
```

深拷贝会复制一份和源数组一样的数组，新数组与源数组不会存放在同一内存位置中，因此，对新数组的修改不会影响源数组。下面的代码演示了 b 使用 copy 方法从源数组 a 复制一份拷贝的情况。可以看到，修改 b 后，a 仍然不变。

```
>>> a=ones((2,3))
>>> d = a.copy()
>>> b[1,2]=2
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
```

2. 矩阵

(1) 创建矩阵。Numpy 的矩阵对象与数组对象相似，主要不同之处在于，矩阵对象的计算遵循矩阵数学运算规律。矩阵使用 matrix 函数创建，以 (2,2) 大小的矩阵（2 行 2 列）为例，可用以下两种方式定义参数：

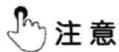
```
'第1行第1列元素 第1行第2列元素；第2行第1列元素 第2行第2列元素'
[[第1行第1列元素, 第1行第2列元素], [第2行第1列元素, 第2行第2列元素]]
下面的代码演示了矩阵的创建及类型查询方法。
```

```
>>> A = matrix('1.0 2.0; 3.0 4.0')###矩阵A
>>> A
[[ 1.  2.]
 [ 3.  4.]]
>>> B = matrix([[1.0,2.0],[3.0,4.0]])###矩阵B
>>> B
matrix([[ 1.,  2.],
       [ 3.,  4.]])
>>> type(A) # 查询A变量的类型
<class 'numpy.matrixlib.defmatrix.matrix'>
```

(2) 矩阵运算。矩阵的常用数学运算有转置、乘法、求逆等。下面的代码演示了矩阵的基本运算。

```
>>> A.T #转置
[[ 1.  3.]
 [ 2.  4.]]
```

```
>>> X = matrix('5.0 7.0')
>>> Y = X.T #转置
>>> Y
[[5.]
 [7.]]
>>> print A*Y #矩阵乘法
[[19.]
 [43.]]
>>> print A.I #逆矩阵
[[-2.  1.]
 [ 1.5 -0.5]]
>>> solve(A, Y) #解线性方程组
matrix([[-3.],
       [ 4.]])
```



注意

关于 Numpy 及其函数的更多信息可查阅 Numpy 官网：

http://wiki.scipy.org/Numpy_Example_List

4.1.3 pylab、matplotlib 绘图

为了验证算法的有效性，机器学习通常需要进行绘图，pylab、matplotlib 等模块是专业的 Python 绘图模块。

1. sin 函数绘制

在二维坐标系中绘图的基本方式是使用 plot 方法，其参数分别为 x 轴数值、y 轴数值，这里的数值可以是单个数也可以是 Numpy 的一维数组对象。下面的代码演示了 sin 函数图像的绘制。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)
```

如图 4-2 所示为 sin 函数图像效果图，从效果图上观察，曲线清晰，坐标系的标尺根据绘制参数已进行自动调整。

2. cos 函数绘制

下面的代码使用 plot 方法绘制 cos 函数图像。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 5, 0.1);
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

绘图效果如图 4-3 所示。

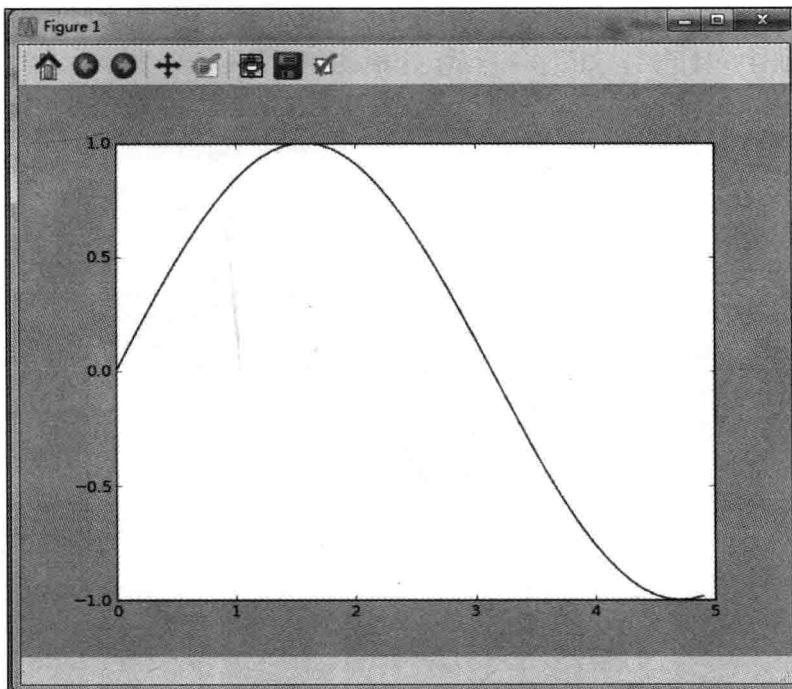


图 4-2 \sin 函数图像效果图

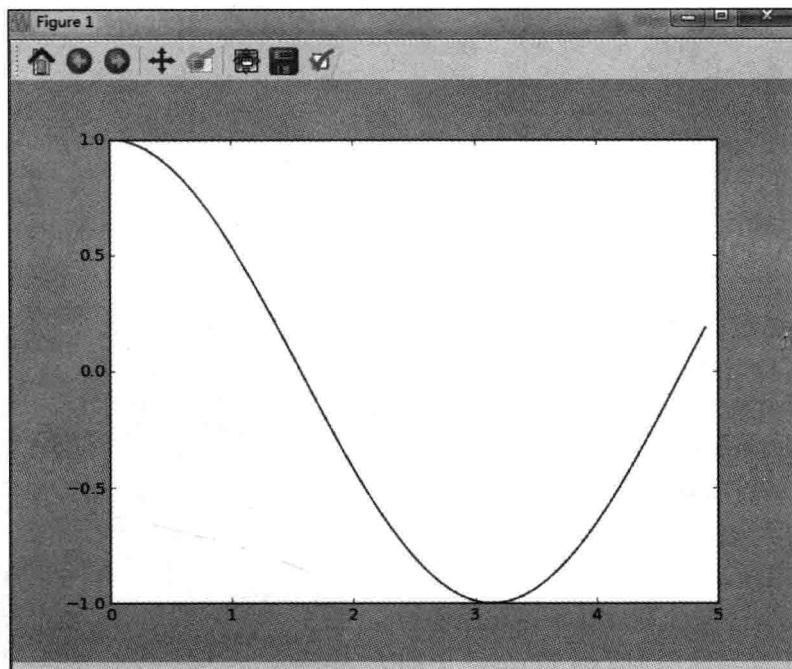


图 4-3 \cos 函数图像效果图

进一步扩充图 4-3 所示 cos 函数绘制范围，将自变量 x 的范围扩大到 $[-8,8]$ ，三角函数 cos 图像的周期性一目了然，如图 4-4 所示。下面是绘制代码。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-8, 8, 0.1);
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

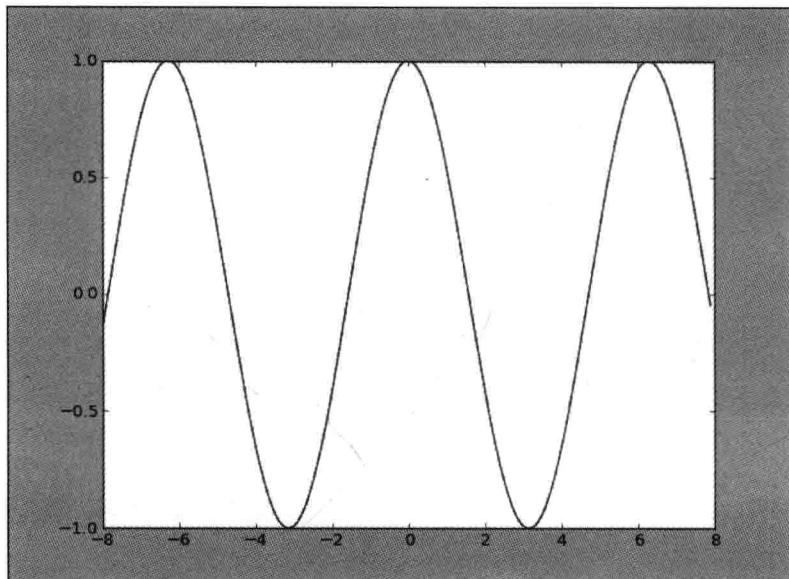


图 4-4 cos 函数周期图像

4.1.4 图像基础

1. 数字图像

数字图像是指将二维图像用有限数字的数值像素表示，像素表面上看起来不像分离的点，但实质它们就是点。例如，图 4-5 所示的树叶写真就是由很多像素点组成的，但用肉眼无法观察到像素点的存在。

我们使用一款取像素的软件对图 4-5 进行分析，如图 4-6 所示。可以看到，线的十字交叉处就是一个像素点，软件显示，在图像的 [459,530] 处像素点的值为 (64,77,67)。中间是放大的这部分图片区域，放大后，图像仿佛



图 4-5 树叶写真

由一个个小的颗粒组成，将这些颗粒进一步放大，就能看到颗粒是由若干个像素点组成，如果将图像完全放大，就能看清每个像素点的存在了。

(64, 77, 67) 就是图 4-6 中某点的像素值。

每个像素点可有各自的颜色值，可采用三原色显示，因而又分成红、绿、蓝三个子像素（RGB 色域），或者青、品红、黄和黑（CMYK 色域），通常计算机的图像采用的像素标准为红、绿、蓝三个子色。图 4-6 所示十字交叉处的像素值含义为：红色值为 64，绿色值为 77，蓝色值为 67。

分辨率是度量图像内数据量多少的一个参数，通常表示成每英寸像素数和每英寸点数。分辨率越高，图像包含的数据越多，就越能表现更丰富的细节，图形文件就越大。从图 4-7 能较直观地看出这个效果，随着分辨率的增加，字母 R 越来越清晰。



图 4-6 树叶放大的颗粒效果（附彩图）

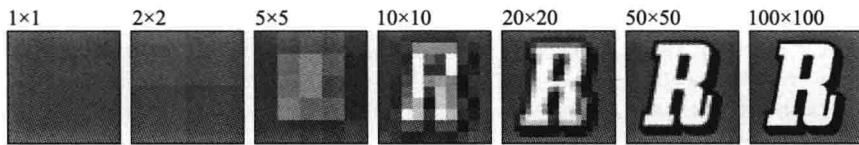


图 4-7 R 字母在不同分辨率下的效果

2. OpenCV 的 Python 绑定库实例

假设图像的分辨率为 800×600 ，则每一条水平线上包含有 800 个像素点，共有 600 条线，在计算机中可以使用一个 800 列 600 行的数组来表示该图像。数组的每个元素就是一个像素点，比如，第 100 行第 200 列的元素就是图像的第 100 条水平线第 200 个像素点的像素值。那么如何提取图像中的像素值呢？

可以使用 OpenCV 的 Python 绑定库完成这些操作。OpenCV 作为跨平台的计算机视觉库，拥有包括 500 多个跨平台图像处理的中、高层 API，对图像像素的读写自然不在话下。

使用 OpenCV 函数库之前，需要先导入其 Python 绑定库。

```
import cv2
```

OpenCV 函数对像素点的读写操作可理解为对图像矩阵的存取，OpenCV 图像矩阵中每个像素点的值由蓝色值、绿色值、红色值 3 个部分组成，三色值组合成一个一维数组。假设 A 图像的高度（行数）为 H，宽度（列数）为 W，则 A 图像对应的图像矩阵大小为 $H \times W \times 3$ ，A 图像矩阵可表示 H 行 W 列（共 $H \times W$ 个）像素点的组合。

如果想读取 A 图像 150 行 20 列处的像素值（设 A 的图像矩阵变量为 img_a），可进行如下访问：

```
img_a[150, 20, :]
```

下面的代码中第 1 行是 150 行 20 列处像素的蓝色值，第 2 行是 150 行 20 列处像素的绿

色值，第3行是150行20列处像素的红色值。

```
blue=img_a[150,20,0]
green=img_a[150,20,1]
red=img_a[150,20,2]
```

下面以几个经典实例演示 OpenCV 的 Python 绑定库的使用方法。

(1) 显示图像。程序原理是，首先使用 imread(文件名)读取图像文件，生成图像矩阵，然后调用 imshow 方法显示图像，调用 waitKey()方法等待按键，最后调用 destroyAllWindows()销毁窗口，这样可方便查看图像。

```
#!/usr/bin/env python
#4-1.py
import cv2
fn="test1.jpg"

if __name__ == '__main__':
    print 'http://blog.csdn.net/myhaspl'
    print 'myhaspl@qq.com'
    print
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    cv2.imshow('preview', img)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

效果如图 4-8 所示。

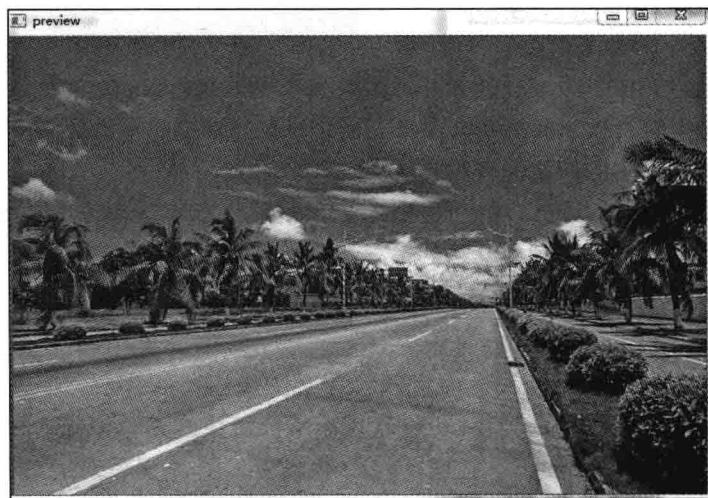


图 4-8 显示图像

(2) 随机生成像素。程序的原理是，首先产生空图像矩阵，然后确定矩阵的 2000 个随机位置，最后将随机位置处的像素值设置为随机数数组。下面是源代码。

```
#!/usr/bin/env python
```

```

#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#随机生成像素点
#4-2.py
#导入numpy和opencv函数库
import numpy as np
import cv2

if __name__ == '__main__':
    #行数
    sz1 = 200
    #列数
    sz2 = 300
    print u'产生空图像矩阵(%d*%d) ...' % (sz1, sz2)
#产生空图像矩阵,大小为sz1*s2(行数*列数),本程序为200*300
    img = np.zeros((sz1, sz2, 3), np.uint8)
    pos1=np.random.randint(200, size=(2000, 1))###行位置随机数组
    pos2=np.random.randint(300, size=(2000, 1))###列位置随机数组
    #在随机位置处设置像素点值
    for i in range(2000):
        img[pos1[i],pos2[i],[0]]=np.random.randint(0,255)
        img[pos1[i],pos2[i],[1]]=np.random.randint(0,255)
        img[pos1[i],pos2[i],[2]]=np.random.randint(0,255)

    #显示图像
    cv2.imshow('preview', img)
    #等待按键
    cv2.waitKey()
    #销毁窗口
    cv2.destroyAllWindows()

```

随机产生像素点后，创建新窗口并显示含彩色雪花点的图像，运行以上代码：

产生空图像矩阵(200*300) ...

效果如图 4-9 所示。

(3) 获取图像大小。程序通过图像矩阵的 shape 属性获取图像大小，shape 返回 tuple 元组，元组的第一个元素为高度，第二个元素为宽度，第 3 个元素为 3（像素值由三原色组成）。

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#4-3.py
import cv2
import numpy as np
fn="test2.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    #获取图像矩阵大小
    sp=img.shape

```



图 4-9 随机产生若干像素点（附彩图）

```

print sp
#高度, 即行数
sz1=sp[0]
#宽度, 即列数
sz2=sp[1]
print 'width:%d\nheight:%d'%(sz2,sz1)

```

运行效果如下, 程序返回图像的高为 435, 宽为 656。

```

loading test1.jpg ...
(435, 656, 3)
width:656
height:435

```

(4) 调节图像亮度。调节的原理是, 将像素值变小, 则将亮度调小, 全部色彩变暗; 将像素值变大, 则将亮度调大, 全部色彩变亮。

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#4-4.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ji=0
    #将全部色彩变暗
    for xi in xrange(0,w):
        for xj in xrange (0,h):
            #将像素值整体减少, 设为原像素值的20%
            img[xj,xi,0]= int(img[xj,xi,0]*0.2)
            img[xj,xi,1]= int(img[xj,xi,1]*0.2)
            img[xj,xi,2]= int(img[xj,xi,2]*0.2)
    #显示进度条
    if xi%10==0 :print '.',
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    cv2.waitKey()
    cv2.destroyAllWindows()
    print ''
    print u'正在处理中',
    #将全部色彩变亮
    for xi in xrange(0,w):
        for xj in xrange (0,h):
            #将像素值整体增加, 设为原像素值的1020%
            img[xj,xi,0]= int(img[xj,xi,0]*10.2)
            img[xj,xi,1]= int(img[xj,xi,1]*10.2)
            img[xj,xi,2]= int(img[xj,xi,2]*10.2)

```

```

if xi%10==0 :print '.',
#显示图像
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

上面程序将图像每个像素值减少，实现图像亮度变暗的效果，如图 4-10 所示。然后每个像素值增大，实现图像亮度变亮的效果。

如图 4-11 所示为图像变亮效果，因为像素值过大，已经出现失真现象。



图 4-10 图像变暗（附彩图）

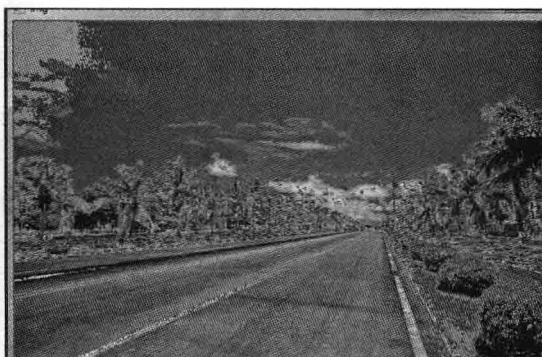


图 4-11 图像变亮（附彩图）

(5) 图像日落效果。日落效果的生成原理很简单，将蓝色值和绿色值设为原来的 70%，红色值不变，设图像矩阵为 img。代码如下：

```

#生成日落效果
for xi in xrange(0,w):
    for xj in xrange (0,h):
        img[xj,xi,0]= int(img[xj,xi,0]*0.7)###蓝色值为原来的70%
        img[xj,xi,1]= int(img[xj,xi,1]*0.7)###绿色值为原来的70%

```

完整代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#4-5.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    #生成日落效果

```

```

for xi in xrange(0,w):
    for xj in xrange (0,h):
        img[xj,xi,0]= int(img[xj,xi,0]*0.7)###蓝色值为原来的70%
        img[xj,xi,1]= int(img[xj,xi,1]*0.7)###绿色值为原来的70%
    #显示进度条
    if xi%10==0 :print '.',
#显示图像
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行效果如图 4-12 所示。

(6) 负片与水印效果。生成负片的原理是，将像素的三色值设为(255-原值)。设图像矩阵为 img，代码如下：

```

#生成负片
b, g, r = cv2.split(img)
b=255-b
g=255-g
r=255-r

```

水印效果的原理是，调用 putText 函数，以图像矩阵为第 1 个参数，输出内容为第 2 个参数，在图像上直接输出水印文字。代码如下：

```

#加上水印
cv2.putText(img,"machine learning", (20,20),cv2.FONT_HERSHEY_PLAIN, 2.0, (0, 0, 0), thickness = 2)
cv2.putText(img,"Support Vector Machines(SVMs) is an algorithm of machine learning.", (20,100),cv2.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 0), thickness = 2)

```

负片与水印效果的完整代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#4-6.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    #读取图像文件
    img = cv2.imread(fn)
    #获取图像大小
    w=img.shape[1]
    h=img.shape[0]
    ii=0

```



图 4-12 图像日落效果（附彩图）

```

#生成负片
b, g, r = cv2.split(img)
b=255-b
g=255-g
r=255-r
#直接通过索引改变色彩分量
img[:, :, 0]=b
img[:, :, 1]=g
img[:, :, 2]=r
#加上水印
cv2.putText(img, "machine learning", (20, 20), cv2.FONT_HERSHEY_PLAIN, 2.0, (0, 0, 0), thickness = 2)
cv2.putText(img, "Support Vector Machines(SVMs) is an algorithm of machine learning.", (20, 100), cv2.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 0), thickness = 2)
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行效果如图 4-13 所示。

(7) 图像平铺。图像平铺的原理是，首先计算平铺后的图像大小，生成同样大小的空白图像，然后在空白图像中逐个像素复制图像，直接将空白图像像素值设置为平铺后该位置对应的像素值，复制的顺序是逐行复制，横向平铺 5 个图像，纵向平铺 2 个图像，最后显示图像效果。下面是完整代码：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-7.py
import cv2
import numpy as np
fn="test.png"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print '正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    #横向平铺5个图像
    sz1=w*5
    #纵向平铺2个图像
    sz0=h*2

```

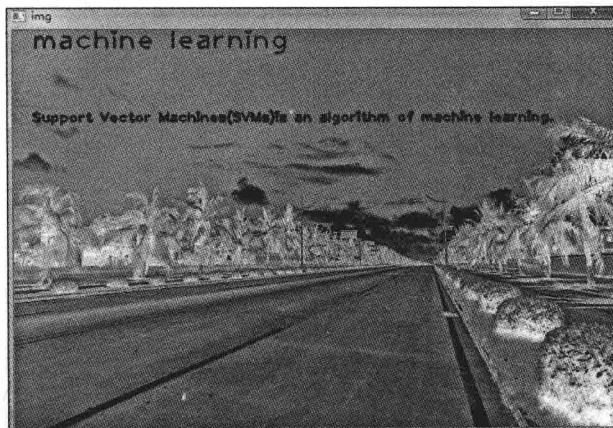


图 4-13 负片和水印效果（附彩图）

```

#创建空白图像，然后将图片排列
myimg1=np.zeros((sz0,sz1,3), np.uint8)
#逐个像素复制
img_x=0
img_y=0
for now_y in xrange(0,sz0):
    #增加行数
    for now_x in xrange(0,sz1):
        #复制对应位置的图像像素点
        myimg1[now_y,now_x,0]=img[img_y,img_x,0]
        myimg1[now_y,now_x,1]=img[img_y,img_x,1]
        myimg1[now_y,now_x,2]=img[img_y,img_x,2]
        #增加列数
        img_x+=1
        if img_x>=w:
            img_x=0
        img_y+=1
        if img_y>=h:
            img_y=0
        print '.',
cv2.namedWindow('img1')
cv2.imshow('img1', myimg1)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行效果如图 4-14 所示。



图 4-14 图像平铺

(8) 旋转并平铺图像。与刚才的例子相似，但多了一步旋转操作。旋转的原理是将图像矩阵转换为它的转置矩阵，旋转算法是将新图像矩阵 [h,w] 处的像素设为原图像矩阵 [w,h] 处的值（这里的值是一维矩阵），相当于矩阵转置的算法。设 myimg1 为图像矩阵，编写代码如下：

```

for now_y in xrange(0,sz0):
    for now_x in xrange(0,sz1):
        myimg1[now_x,now_y,:]=img[img_y,img_x,:]

```

旋转并平铺图像完整代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-8.py
import cv2
import numpy as np
fn="test.png"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print 'working',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    #w为宽度, h为高度
    sz1=w*2
    sz0=h*3
    #创建空白图像, 然后将图片排列
    myimg1=np.zeros((sz1,sz0,3), np.uint8)
    #翻转并生成图像
    #逐个复制像素
    img_x=0
    img_y=0
    for now_y in xrange(0,sz0):
        for now_x in xrange(0,sz1):
            #旋转图像
            myimg1[now_x,now_y,:]=img[img_y,img_x,:]
            img_x+=1
            #新的一次平铺
            if img_x>=w:
                img_x=0
            img_y+=1
            #新的一次平铺
            if img_y>=h:
                img_y=0
            print '.',
    cv2.namedWindow('img1')
    cv2.imshow('img1', myimg1)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

运行效果如图 4-15 所示。

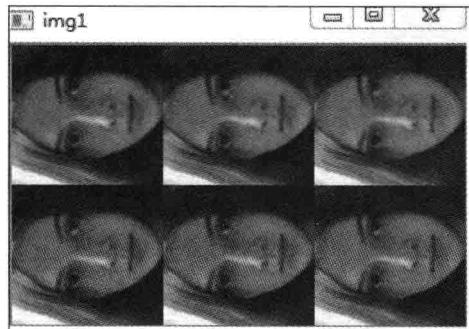


图 4-15 图像旋转和平铺

4.1.5 图像融合与图像镜像

本节的两个例子是对上节内容精华的总结，较好地综合了 OpenCV 的基础功能。

1. 图像融合

图像融合的原理是，让新图像的每个像素成为两个源图像中对应像素的平均值之和，

即：将两个图像的像素值取 50% 后相加。为简化计算，直接选取其中一个源图像作为新图像，设新图像矩阵为 myimg2，编写代码如下：

```
#每个像素为2个像素的平均值
for y in xrange(0,sz0):
    for x in xrange(0,sz1):
        myimg2[y,x,:]=myimg1[y,x,:]*0.5+myimg2[y,x,:]*0.5
```

完整代码如下（关键之处有注释）：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-9.py
import cv2
import numpy as np
fn1="he1.jpg"
fn2="he2.jpg"
if __name__ == '__main__':
    print 'working',
    myimg1 = cv2.imread(fn1)
    myimg2 = cv2.imread(fn2)
    #取得图像大小
    w=myimg1.shape[1]
    h=myimg1.shape[0]
    sz1=w
    sz0=h

    #每个像素为2个像素的平均值之和，进行图像融合
    for y in xrange(0,sz0):
        for x in xrange(0,sz1):
            myimg2[y,x,:]=myimg1[y,x,:]*0.5+myimg2[y,
            x,:]*0.5
            print '.',

    cv2.namedWindow('img2')
    cv2.imshow('img2', myimg2)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

效果如图 4-16 所示。



图 4-16 图像融合

2. 图像镜像

图像纵向镜像的原理是，首先获取图像的宽度，将宽度的 50% 取整后作为图像的纵向中线；然后以中线为轴，将图像左边反向复制到图像右边，使图像最右边一列的像素点等于图像最左边一列的像素点。比如，图像大小为 200×300 （高 200，宽 300），第 180 行 170 列（索引为 [180,170,:]）的像素点值为第 180 行第 130 列的像素点值 ($300-170=130$)。

横向镜像与纵向镜像类似，不同之处在于将高度的 50% 取整后作为图像的横向中线，复制时是最下边一行的像素点值等于最上边一行的像素点值。

纵向镜像可按如下形式编写代码：

```
#纵向生成镜像
mirror_w=w/2
for j in xrange(0,h):
    for i in xrange(0,mirror_w):
        img[j,i,:]=img[j,w-i-1,:]
```

下面的代码演示了图像的纵向镜像。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-10.py
import cv2
import numpy as np

fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print '正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    #纵向生成镜像
    mirror_w=w/2
    for j in xrange(0,h):
        for i in xrange(0,mirror_w):
            img[j,i,:]=img[j,w-i-1,:]
        print '.',
    #显示图像
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    cv2.waitKey()
    cv2.destroyAllWindows()
```



运行效果如图 4-17 所示。

图 4-17 图像镜像

4.1.6 图像灰度化与图像加噪

1. 图像灰度化

图像灰度化的原理是，彩色图像中的每个像素的颜色由 R、G、B 三个分量决定，而每个分量的取值范围为 0~255。而灰度图像是 R、G、B 三个分量相同的一种特殊的彩色图像，其算法有以下两种：

- (1) 求出每个像素点的 R、G、B 三个分量的平均值，然后将这个平均值赋予给这个像素的三个分量。
- (2) 根据 RGB 和 YUV 颜色空间的变化关系，建立亮度 Y 与 R、G、B 三个颜色分量的

对应关系： $Y=0.3R+0.59G+0.11B$ ，以这个亮度值表达图像的灰度值。

OpenCV 有相关的函数 cvtColor，用它可直接完成灰度化操作。设 img 为源图像矩阵，myimg1 为灰度化后的目标图像矩阵，编写代码如下：

```
#复制并转换为灰度化图像
myimg1=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

下面的代码演示了图像的复制与图像的灰度化操作。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#4-11.py
import cv2
import numpy as np

fn="test2.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    sp=img.shape
    print sp
    #获取图像大小
    #height
    sz1=sp[0]
    #width
    sz2=sp[1]
    #显示图像大小
    print 'width:%d\nheight:%d'%(sz2,sz1)
    #创建一个窗口并显示图像
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    #复制图像矩阵，生成与源图像一样的图像，并显示
    myimg2= img.copy();
    cv2.namedWindow('myimg2')
    cv2.imshow('myimg2', myimg2)

    #复制并转换为灰度化图像，并显示
    myimg1=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.namedWindow('myimg1')
    cv2.imshow('myimg1', myimg1)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

上面的代码生成了与源图像一样的新图像，并生成了另一个源图像的灰度化图像，运行效果如图 4-18 所示。

现在大部分的彩色图像都是采用 RGB 颜色模式，处理图像的时候，要分别对

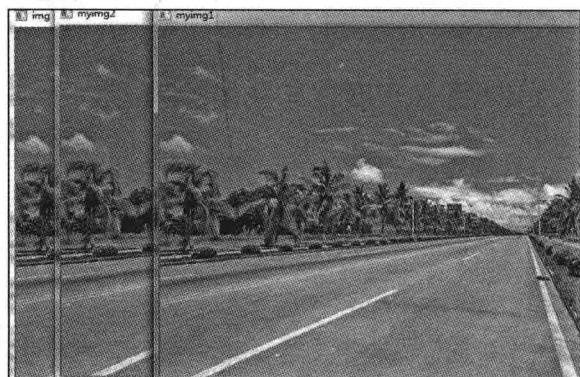


图 4-18 图像灰度化（附彩图）

RGB 三种分量进行处理。实际上 RGB 并不能反映图像的形态特征，只是从光学的原理进行颜色的调配。把图像转换成 8 位的灰度值图像直接进行处理，可以通过直方图、灰度变化及正交变换之类数学运算对图像做进一步处理，比如说图像识别等。如果有必要，可将图像二值化，这样有利于对图像进一步处理，使图像数据量减小，突显出感兴趣的目标的轮廓。如图 4-19 所示为某汽车图像二值化的效果。



图 4-19 图像二值化

2. 图像加噪

给图像人为加噪的原理是，将图像若干个像素点的值设为噪声点的值。比如，为图像加上很多像素值为 [25,20,20] 的像素点，编写代码如下：

```
for k in xrange(0, coutn):
    xi = int(np.random.uniform(0, img.shape[1]))
    xj = int(np.random.uniform(0, img.shape[0]))
    if img.ndim == 2:
        # 灰度图像
        img[xj, xi] = 255
    elif img.ndim == 3:
        # 非灰度图像, 图像加噪
        img[xj, xi, 0] = 25
        img[xj, xi, 1] = 20
        img[xj, xi, 2] = 20
```

上面的代码对 `img.ndim` 进行判断的用意在于，如果图像是灰度化图像，则 `img.ndim` 为 2，灰度化图像的像素值不存在红、绿、蓝三色之分，仅有灰度值，因此像素值仅需要一个，将对应噪声点位置的值设为 255 即可。

下面的代码演示了图像加噪的算法，为彩色图像人为加上 100000 个色彩随机的噪声点。

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#4-12.py
import cv2
import numpy as np
# 需要加噪的图像文件名
fn="test1.jpg"
if __name__ == '__main__':
    # 加载图像
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    # 噪声点数量
    coutn=100000
    for k in xrange(0, coutn):
        # 获取图像噪声点的随机位置
        xi = int(np.random.uniform(0, img.shape[1]))
        xj = int(np.random.uniform(0, img.shape[0]))
        # 图像加噪声点
```

```

if img.ndim == 2:
    img[xj,xi] = 255
elif img.ndim == 3:
    img[xj,xi,0]= 25
    img[xj,xi,1]= 20
    img[xj,xi,2]= 20
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行效果如图 4-20 所示。



图 4-20 图像加噪

上述程序的运行原理是将图像数据矩阵随机位置的像素点设为(25,20,20)，当随机的像素点数量较大时，就在图像上生成了噪声。

加上噪声的图像是为了实验图像识别的效果，有些机器学习算法对没有噪声的图像识别的效果很好，但如图 4-20 这种噪声较多的情况效果就很不理想了，因为在实际工程应用中，很难保证采集到的图像清晰可靠，所以需要人为给图像加上噪声，以方便后期对算法效果进行验证。

4.1.7 声音基础

1. 声音原理

声音是由物体的机械振动形成的，发生声音的振动源叫作“声源”。振动着的鼓皮、琴弦、扬声器等都是声源，人的声带也是声源。声音必须通过媒质才能传播，空气、水、金属、木材等是最常见的媒质。声波的频率是每秒钟往复振动的次数，一来一往为一次，又称

一周，声波的频率也就是声音的频率，频率单位为赫兹（Hz），每秒振动一周为1Hz。“波长”是声源每振动一周声波所传播的距离，频率越高则波长越短，波长同频率成反比。

“相位”可简称为“相”。一般地说，相位是用来描述简谐振动的，在一个周波之内，任何一点的“相”都不相同，各对应于一个确定的相位角值；而在另一个周波，各种相位将会重复出现。所以在声波传播的路径上，每隔一个波长的距离，其相位相同。

声音的音调是由它的基频决定的，基频越高则音调也越高。如在音乐中中央C的基频是261.6Hz，而A调的基频则是440Hz。通常将声音分为以下频带：20Hz、25Hz、31.5Hz、40Hz、50Hz、63Hz、80Hz、100Hz、125Hz、160Hz、200Hz、250Hz、315Hz、400Hz、500Hz、630Hz、800Hz、1kHz、1.25kHz、1.60kHz、2.0kHz、2.5kHz、3.15kHz、4.0kHz、5.0kHz、6.3kHz、8.0kHz、10kHz、12.5kHz、16kHz、20kHz。一般来说，人耳可感受的正弦波的范围是从20Hz的低频声音到20kHz的高频声。

2. 声音波形

声音波波形属于正弦波，拥有振幅和频率两个特征，振幅就是音量，频率就是音调。下面调用Python的WAV声音处理库以及Numpy科学计算库显示一段声音的波形。

显示声音波形数据的主要步骤如下：

(1) 打开WAV文件，使用wave库的open方法，主要参数为文件名和存取文件方式。

```
#以读方式打开WAV文档
f = wave.open(r"back.wav", "rb")
```

(2) 读取格式信息，使用wave库的getparams方法。该方法返回的信息中比较重要的是前4项，依次为通道数、样本宽度、样本频率、波形数据长度。

```
#读取格式信息
#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
```

(3) 读取波形数据，波形数据是WAV文件采样后生成的采样数据，使用wave库的readframes方法读取，该方法返回的数据是字符类型。

```
#读取波形数据
str_data = f.readframes(nframes)
```

(4) 转换波形数据为Numpy的整型数组对象。

```
#将波形数据转换为数组
wave_data = np.fromstring(str_data, dtype=np.short)
wave_data.shape = -1, 2
wave_data = wave_data.T
```

(5) 计算时间轴。

```
time = np.arange(0, nframes) * (1.0 / framerate)
```

(6) 绘制波形，绘制前调用pylab的subplot方法创建两个上下形式的绘图区，每个绘图区各绘制一个声道的数据。下面程序中subplot方法的参数共3位整数，从左边开始每位

依次表示绘图区总数、列数、创建区域所属绘图的索引，比如 subplot(212) 表示绘图区有 2 个，一共 1 列，当前索引为第 2 个绘图区。

```
#绘制波形
#创建上边的绘图区
pl.subplot(211)
#绘制左声道
pl.plot(time, wave_data[0])
#创建下边的绘图区
pl.subplot(212)
#绘制右声道
pl.plot(time, wave_data[1], c="g")
```

上述绘制声音波形过程的完整代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-13.py
import wave
import pylab as pl
import numpy as np
print 'working...'
#打开WAV文档
f = wave.open(r"back.wav", "rb")
#读取格式信息
#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
#读取波形数据
str_data = f.readframes(nframes)
f.close()
#将波形数据转换为数组
wave_data = np.fromstring(str_data, dtype=np.short)
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
#绘制波形
pl.subplot(211)
pl.plot(time, wave_data[0])
pl.subplot(212)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()
```

程序读取声音文件后，绘制出如图 4-21 所示的波形。

这个波形表现出：声音信号较连续，随着时间的推移，变化不明显，没有停顿，因此，这是一段音乐或噪声等声音而不是人声，因为人说话的声音有个特点，就是每个字之间有少量停顿。语音停顿期间，声音采样软件采样不到数据，过了这个停顿期，波形会有明显的变化，如图 4-22 所示波形就是典型的说话声音波形。

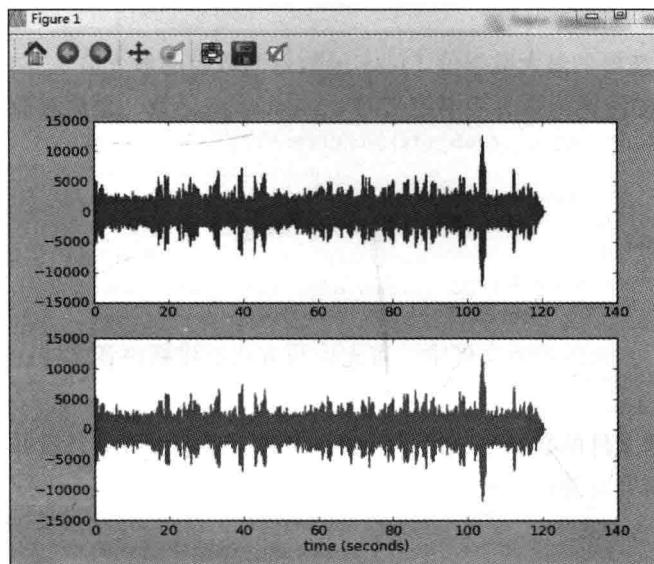


图 4-21 声音波形绘制

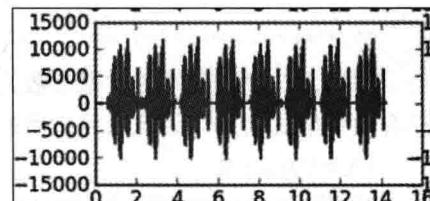


图 4-22 说话声音波形

4.1.8 声音音量调节

声音音量的调节方式与图像亮度调整类似，不同的是音量调节的是波形大小。音量调节通过调节采样波形的大小实现，采样数据变大时，声音音量放大，采样数据变小时，声音音量降低。音量不能无限调节，音量过大或过小，会形成难听的噪音，使声音失真。

1. 放大音量

下面编写代码演示音量的放大。为保证声音质量，需要对音量调节范围设置上限和下限（以原声音为基准计算上限、下限）。为此，编写 `wavechange` 函数，计算调整后的数据，其参数 `x` 为每次采样的波形数据，`dwmax` 为上限，`dwmin` 为下限，该函数仅会将上限与下限之间区域内的数据放大为原来的 1.5 倍，在此区域外的数据则设置为上限或下限。

```
def wavechange(x,dwmax,dwmin):
    if x!=0:
        if abs(x)>dwmax:
            x=x/abs(x)*dwmax
        elif abs(x)<dwmin:
            x=x/abs(x)*dwmin
        else:
            x=x*1.5
    return x
```

为保证放大后声音不失真，可采用以原声音为基准的放大策略，声音波形图像类似正弦函数图像，在以时间轴为 X 轴、采样数据为 Y 轴的坐标系中，波形数据可正可负，上下波动。因此，以原声音数据的最大值为依据计算上下限，上限为原声音数据最大值的 88%，

下限为原声音数据最大值的 14%。

使用 wave_data.max() 获取原声音波形的最大数据值 (max 函数返回数组的最大值)，然后通过 frompyfunc 函数设置调节音量的回调函数为刚刚定义的 wavechange 函数，最后对数据进行放大调节。

```
#放大音量
change_dwmax=wave_data.max()/100*88
change_dwmin=wave_data.max()/100*14
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
```

声音数据放大后，需要将新数据写入新的声音文件中。首先以写方式新建新声音文件：

```
fo = wave.open(r"jg.wav", "wb")
```

然后，设置新文件的数据参数为源文件的数据参数，并写到新声音文件中。放大音量并没有改变格式信息，因此，放大后的声音与源声音的格式信息一样。

```
#写波形数据参数
print "save new wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
```

最后调用 writeframes() 方法，以放大后声音数据为参数将数据写入新建的声音文件中。

```
fo.writeframes(new_str_data)
```

下面的代码演示了放大音量的算法，并绘制出了源声音波形与放大后的声音波形。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-14.py
import wave
import pylab as pl
import numpy as np
def wavechange(x,dwmax,dwmin):
    if x!=0:
        if abs(x)>dwmax:
            x=x/abs(x)*dwmax
        elif abs(x)<dwmin:
            x=x/abs(x)*dwmin
        else:
            x=x*1.5
    return x
#打开WAV文档
f = wave.open(r"speak.wav", "rb")
fo = wave.open(r"jg.wav", "wb")
#读取波形数据
#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
```

```

print "read wav data...."
str_data = f.readframes(nframes)
#将波形数据转换为数组，并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
#放大音量
change_dwmax=wave_data.max()/100*88
change_dwmin=wave_data.max()/100*14
wave_change = np.frompyfunc(wavechange, 3, 1)
new_wave_data = wave_change(wave_data, change_dwmax, change_dwmin)
new_wave_data = new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
#写波形数据参数
print "save new wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
#绘制源声音波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(221)
pl.plot(time, wave_data[0])
pl.subplot(222)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
#绘制放大音量波形
new_wave_data.shape = -1, 2
new_wave_data = new_wave_data.T
new_time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(223)
pl.plot(new_time,new_wave_data[0])
pl.subplot(224)
pl.plot(new_time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

如图 4-23 所示波形图中，上部为源声音，下部显示了音量放大后的波形。下部的波形数据范围为 -20000~20000，而上部范围为 -15000~15000，下部波形整体比上部大很多。下载本书的代码包运行后，可听到音量放大后的声音效果。

2. 降低音量

音量降低可通过将采样波形变小来实现，具体来说，就是把每个采样数据按指定比例缩小，同时将缩小幅度控制在合理的范围内，保证音量降低后声音仍然清晰。

(1) 根据上下限参数对波形数据进行调节，定义缩小波形数据的函数为 wavechange。

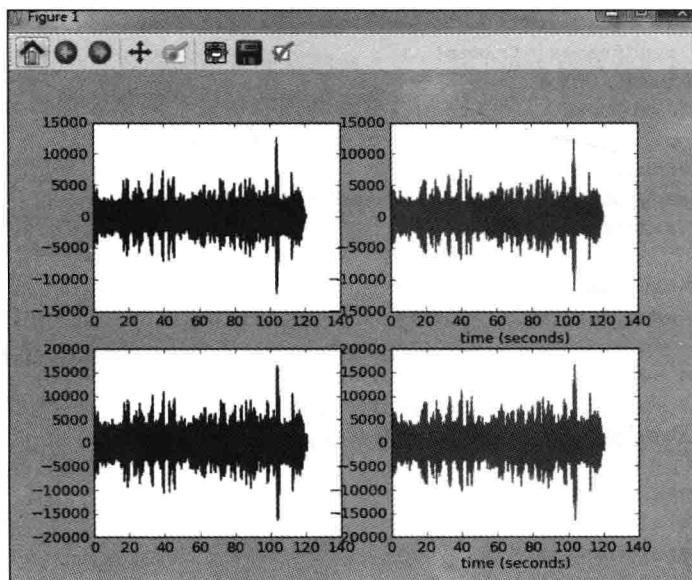


图 4-23 声音音量放大波形

```
def wavechange(x,dwmax,dwmin):
    if x!=0:
        if abs(x)<dwmax and abs(x)>dwmin:
            x=x*0.5
        else:
            x=x*0.2
    return x
```

(2) 与放大音量类似,以源声音波形数据的最大值为基准,计算上限和下限,以 `wavechange` 为回调函数,降低音量。

```
#降低音量
change_dwmax=wave_data.max()/100*1
change_dwmin=wave_data.max()/100*0.5
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
```

(3) 生成新波形数据。

```
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
```

(4) 将数据写到新声音文件。

```
#写波形数据参数
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
```

下面的代码演示了降低音量算法。

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-15.py
import wave
import pylab as pl
import numpy as np
print 'working...'
def wavechange(x,dwmax,dwmin):
    if x!=0:
        if abs(x)<dwmax and abs(x)>dwmin:
            x=x*0.5
        else:
            x=x*0.2
    return x
#打开WAV文档
f = wave.open(r"back.wav", "rb")
fo = wave.open(r"jg.wav", "wb")
#读取波形数据
#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
print "read wav data...."
str_data = f.readframes(nframes)
#将波形数据转换为数组，并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
#降低音量
change_dwmax=wave_data.max()/100*1
change_dwmin=wave_data.max()/100*0.5
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
#写波形数据参数
print "save new wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
#绘制源声音波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(221)
pl.plot(time, wave_data[0])
pl.subplot(222)

```

```

pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
#绘制降低音量波形
new_wave_data.shape = -1, 2
new_wave_data = new_wave_data.T
new_time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(223)
pl.plot(new_time,new_wave_data[0])
pl.subplot(224)
pl.plot(new_time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

如图 4-24 所示的波形图中，上面为源声音，下面为降低音量后的声音波形。可明显看出，音量缩小后，其波形幅度为 $-3000 \sim 3000$ ，而源声音波形范围大很多，为 $-15000 \sim 15000$ 。

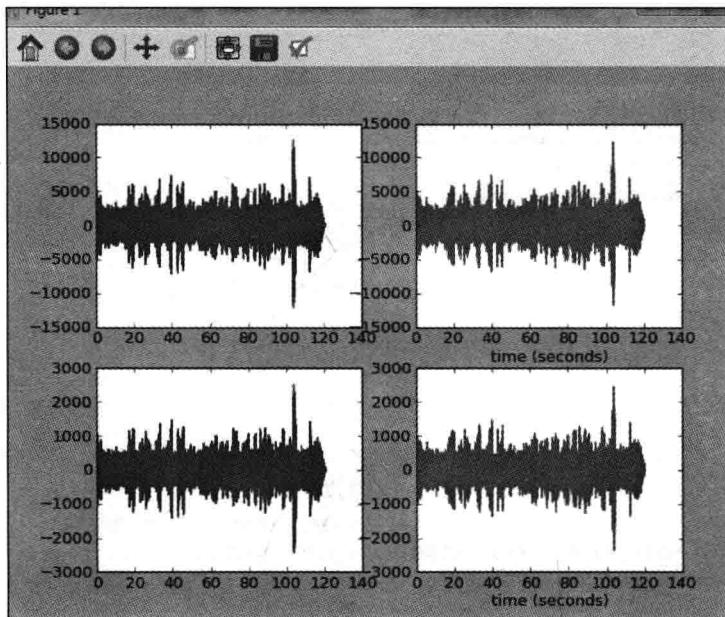


图 4-24 声音音量缩小波形

4.1.9 图像信息隐藏

1. 图像隐藏原理

信息隐藏是不让除预期接收者之外的任何人知晓信息的传递事件或者信息的内容，载体文件相对隐秘文件的大小越大，隐藏后者就越加容易。因此，数字图像在互联网和其他传媒上被广泛用于隐藏消息。

本节讲述的图像隐藏原理是：首先从源图中提取文字图像信息，并记录这个文字图像信息像素点在图像矩阵中的位置；然后，对载体文件进行预处理，将蓝色像素值全部设为偶数；最后，将记录的文字信息像素点在载体文件对应位置的蓝色像素值设为奇数。解密信息是隐藏信息的逆过程，其过程比较简单，即提取载体文件中蓝色像素值为奇数的像素点，将空白图像中这些像素点对应的位置赋予统一的着色。

2. 图像隐藏实例

下面用实例来讲解图像信息隐藏技术。我们的目标是：将如图 4-25 所示的文字隐藏在如图 4-26 所示的载体图片里。要求隐藏后，无法察觉图中隐藏了信息。



图 4-25 含有待隐藏文字的图像



图 4-26 载体图像

本实例隐藏信息的主要过程如下：

(1) 读取源图像（将写上需隐藏文字的信息）和载体图像，构造图像矩阵。

```
img1 = cv2.imread(fn1)
img2 = cv2.imread(fn2)
w=img1.shape[1]
h=img1.shape[0]
```

(2) 在源图像中加上水印文字作为待隐藏文字。

```
#加上需要隐藏的消息
cv2.putText(img1,"hello,world!", (20,300),cv2.FONT_HERSHEY_PLAIN, 3.0,
redcolor, thickness = 2)
cv2.putText(img1,"code by myhaspl:myhaspl@qq.com", (20,60),cv2.FONT_HERSHEY_PLAIN, 2.0, redcolor, thickness = 2)
cv2.putText(img1,"Installing Python is generally easy. ", (1,90),cv2.FONT_HERSHEY_PLAIN, 2, redcolor, thickness = 1)
```

(3) 处理隐藏载体图，将所有蓝色值变成偶数，以便加入隐藏信息。

```
#处理隐藏载体图
#将所有蓝色值变成偶数
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img2[j,i,0]%2)==1:
            img2[j,i,0]=img2[j,i,0]-1
```

(4) 读取源图，将源图的文字像素点在载体文件的对应位置的蓝色像素值设为奇数，将需要隐藏的信息写入目标载体图。

```
#读取源图，并将信息写入目标载体图
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img1[j,i,0],img1[j,i,1],img1[j,i,2]) == redcolor:
            img2[j,i,0] = img2[j,i,0]+1
```

(5) 保存修改后的目标载体图。

```
cv2.imshow('img2-2', img2)
cv2.imwrite(fn3, img2)
```

下面的代码演示了隐藏信息的过程。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#4-16.py
import cv2
import numpy as np
#含有文字的图像
fn1="test1.jpg"
#载体文件
fn2="test2.jpg"
#包含隐藏信息的载体文件
fn3="secret.png"
redcolor=(0, 0, 255)
if __name__ == '__main__':
    print u'正在处理中',
    #图像大小
    img1 = cv2.imread(fn1)
    img2 = cv2.imread(fn2)
    w=img1.shape[1]
    h=img1.shape[0]
    #加上需要隐藏的消息
    cv2.putText(img1,"hello,world!", (20,300),cv2.FONT_HERSHEY_PLAIN, 3.0,
    redcolor, thickness = 2)
    cv2.putText(img1,"code by myhaspl:myhaspl@qq.com", (20,60),cv2.FONT_HERSHEY_PLAIN, 2.0, redcolor, thickness = 2)
    cv2.putText(img1,"Installing Python is generally easy. ", (1,90),cv2.FONT_HERSHEY_PLAIN, 2, redcolor, thickness = 1)

    cv2.namedWindow('img1')
    cv2.imshow('img1', img1)
    cv2.namedWindow('img2-1')
    cv2.imshow('img2-1', img2)
    #处理隐藏载体图
    #将所有蓝色值变成偶数
    for j in xrange(0,h):
        for i in xrange(0,w):
```

```

if (img2[j,i,0] % 2) == 1:
    img2[j,i,0] = img2[j,i,0] - 1
print '.',
mirror_w = w / 2
#读取源图，并将信息写入目标图，将有信息的像素点的蓝色值设为奇数
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img1[j,i,0],img1[j,i,1],img1[j,i,2]) == redcolor:
            img2[j,i,0] = img2[j,i,0] + 1
    print '.',
#保存修改后的目标图，并显示
cv2.namedWindow('img2-2')
cv2.imshow('img2-2', img2)
cv2.imwrite(fn3, img2)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行上段代码将信息隐藏后，肉眼观察载体图像，仍无法察觉与相比之前有任何变化。

下面来看看解密信息过程。解密信息与隐藏信息相反，是隐藏信息的逆过程，主要步骤如下：

(1) 读取载体文件及其大小信息。

```

img = cv2.imread(fn)
w = img.shape[1]
h = img.shape[0]

```

(2) 生成空白图像矩阵，以便绘制解密文字。

```
imginfo = np.zeros((h,w,3), np.uint8)
```

(3) 绘制解密的水印文字。如果蓝色值为奇数，则该像素点为文字。

```

for j in xrange(0,h):
    for i in xrange(0,w):
        if (img[j,i,0] % 2) == 1:
            imginfo[j,i,1] = 255

```

(4) 显示隐藏信息。

```

cv2.imshow('info', imginfo)
cv2.imwrite(fn, imginfo)

```

下面代码演示了解密信息的过程。

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#解密文件
#4-17.py
import cv2
import numpy as np
fn="secret.png"
if __name__ == '__main__':
    print 'loading ...'

```

```

print u'正在处理中',
img = cv2.imread(fn)
w=img.shape[1]
h=img.shape[0]
imginfo = np.zeros((h,w,3), np.uint8)
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img[j,i,0] % 2) == 1:
            #如果蓝色值为奇数，则该像素点为文字
            imginfo[j,i,1]=255
print '.',
cv2.imshow('info', imginfo)
cv2.imwrite(fn, imginfo)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行解密代码，从载体文件中提取信息，效果如图 4-27 所示。

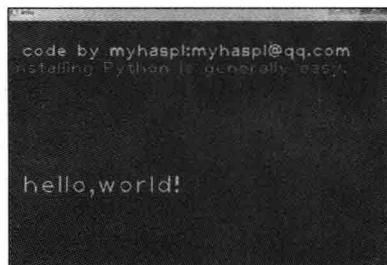


图 4-27 解密后的文字

4.1.10 声音信息隐藏

1. 声音信息隐藏原理

声音文件是一个不错的 информации载体，声音文件数据量大，能隐藏信息的容量也大，假设每秒采集 44100 次，如果所有采样数据全部利用上，每秒的声音可以存储 44100 字节的数据，不过这样达不到信息隐藏的效果，只能利用其中一部分采样数据来存储信息，占有的采样数据越少，信息隐藏效果就越好。

比如，如图 4-28 所示的波形是一段音乐的声音波形，假设某个采样点的数据实际是信息中一个字节大小的数据，那么将这些字节解密后，能还原成一段信息。这种载体的隐藏信息的效果比图像好，一般很难被人发现。

这里采用的隐藏策略是：产生一段正弦波的噪声，然后，在这段噪声中隐藏一段文本文件的内容。下面以实例来讲解这个过程，我们的目标是：将本章前面讲述的 Python 代码文件 4-1.py 隐藏到一段噪声中，解密者如果不知道信息解密的规律，就无法从噪声文件还原这个 Python 代码文件。

2. 声音信息隐藏实例

隐藏信息的具体过程如下：

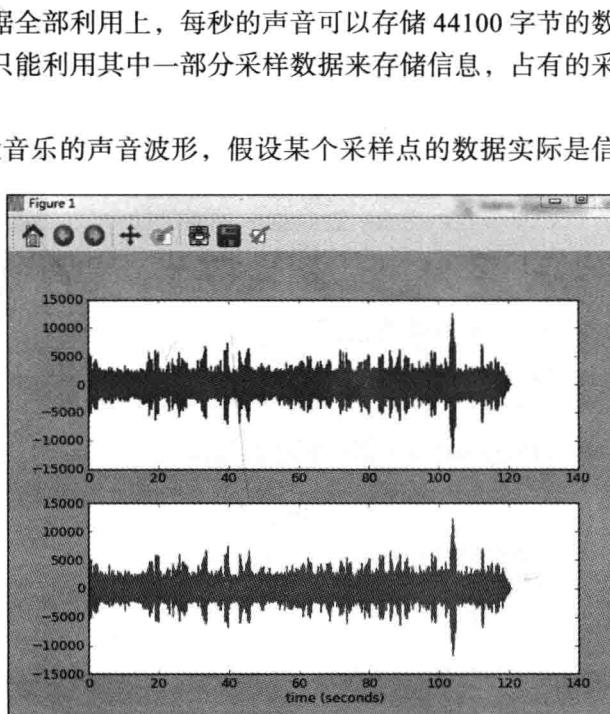


图 4-28 音乐的声音波形

(1) 读取需要隐藏的文本文件，提取其中的文字信息。

```
# 打开文档
fo = wave.open(r"pltest.wav", "wb")
file_object = open('4-1.py')
try:
    all_the_text = file_object.read()
finally:
    file_object.close()
```

(2) 将文字转化为对应的内部编码（本例的文字为英文字母和符号，因此转换为 ASCII 码）。

```
wdata=map(ord,all_the_text)
wdata=np.array(wdata)
```

(3) 设置噪声载体文件的波形参数。载体文件是程序人为生成，所以将幅度设置为适合的区域，为使载体噪声更接近于自然的噪声，将振幅范围设置为 -25600~25600。

```
# 设置波形参数
#采样率
framerate = 44100
#声道数
nchannels=2
#每位宽度
sampwidth=2
#长度
nframes = framerate*4
#振幅
base_amplitude = 200
max_amplitude=128*base_amplitude
```

(4) 计算每个字符的间隔，需要隐藏的若干个字符以等间隔的形式分散在噪声数据中，即：在噪声波形数据中，每隔指定的间隔存放一个字符。

```
#每个字符的间隔
interval=(nframes-10)/lwdata
```

(5) 生成空波形数据，以便写入噪声数据和字符信息。

```
#每周期样本数
wave_data=np.zeros((nframes), dtype=np.short)
```

(6) 生成噪声数据，并将隐藏文字的字符写入噪声数据中，这一步是关键，也是算法的核心。算法把整个采样的线性区域分为两类，如果当前采样时间处于第 4 步计算的间隔处，则表示此处为加密字符区（每个字符区只能存放一个字符），写入的数据为经过加密的字符，在间隔之前的时间区域则为随机噪声区。

算法判断是否到了指定的间隔处，间隔处是字符区，否则是噪声区。如果是噪声区，则随机生成噪声；如果是字符区，则将字符进行加密，写入字符区。此处使用了简单的加密算法（实际应用可使用高强度的加密算法），加密方式为：将字符的 ASCII 码乘以指定的整数后，减去 64 与该整数的乘积。生成的信息隐藏格式如表 4-2 所示。

表 4-2 信息隐藏格式

区域	随机噪声区	加密字符区	随机噪声区	加密字符区
长度	interval	1	interval	1	

算法代码如下：

```
#写噪声数据和隐藏文字的字符
myrand=np.random.rand(nframes)
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        #将隐藏文字的字符通过一定的变化写入噪声数据中
        possamp=wdata[count]*base_amplitude-64*base_amplitude
        count+=1
    elif curpos%60==0:
        #生成随机噪声数据。
        possamp=int(myrand[curpos]*max_amplitude-max_amplitude/2)
    else:
        possamp=0
    wave_data[curpos]=possamp
```

(7) 写波形数据。

```
#写波形数据参数
print "save new wav files...."
str_data=wave_data.tostring()
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.setnframes(nframes)
fo.writeframes(str_data)
```

下面来看看解码信息过程，解码信息是隐藏信息的逆算法，主要步骤如下：

(1) 读取噪声载体文件以及相关格式信息。

```
new_wdata=[]
print u'正在从声音解码文件'
fi = wave.open(r"plttest.wav", "rb")
fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
```

(2) 找到字符区，将其中的字符解密并还原成字符串。

```
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=(fi_wave_data[curpos]+64*base_amplitude)/base_amplitude
        new_wdata.append(possamp)
        count+=1
```

(3) 整理还原字符串，将它们写入文件。

```
my_the_text="".join(map(chr,new_wdata))
```

```
fmy_the_text="".join(map(chr,new_wdata))
file_object = open('mytext.txt', 'w')
file_object.write(my_the_text)
file_object.close()
```

下面程序读取名为 4-1.py 的 Python 源程序文件，将该文本隐藏在声音文件中，然后打开载体声音文件，将文本还原为 Python 程序文件。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#将文件隐藏在声音之中
#4-18.py
import wave
import pylab as pl
import numpy as np
#编码
print u'正在将文件编码进声音'
print "generate wav data...."
#打开文档
fo = wave.open(r"pltest.wav", "wb")
file_object = open('4-1.py')
try:
    all_the_text = file_object.read()
finally:
    file_object.close()
wdata=map(ord,all_the_text)
wdata=np.array(wdata)
lwdata=len(wdata)
#设置波形参数
#采样率
framerate = 44100
#声道数
nchannels=2
#每位宽度
sampwidth=2
#长度
nframes = framerate*4
#振幅
base_amplitude = 200
max_amplitude=128*base_amplitude
#每个字符的间隔
interval=(nframes-10)/lwdata
#每周期样本数
wave_data=np.zeros((nframes), dtype=np.short)
count=0
#写噪声数据和隐藏文字的字符
myrand=np.random.rand(nframes)
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=wdata[count]*base_amplitude-64*base_amplitude
```

```

        count+=1
    elif curpos%60==0:
        possamp=int(myrand[curpos]*max_amplitude-max_amplitude/2)
    else:
        possamp=0
    wave_data[curpos]=possamp
#写波形数据参数
print "save new wav files...."
str_data=wave_data.tostring()
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.setnframes(nframes)
fo.writeframes(str_data)
fo.close()
#绘制波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes/2)
pl.subplot(211)
pl.plot(time, wave_data[0], c="r")
pl.subplot(212)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
#解码
new_wdata=[]
print u'正在从声音解码文件'
fi = wave.open(r"pltest.wav", "rb")
fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
count=0
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=(fi_wave_data[curpos]+64*base_amplitude)/base_amplitude
        new_wdata.append(possamp)
        count+=1
my_the_text="".join(map(chr,new_wdata))
file_object = open('mytext.txt', 'w')
file_object.write(my_the_text)
file_object.close( )
pl.show()

```

运行这段代码，程序输出了编码和解码过程。

```

正在将文件编码进声音
generate wav data....
save new wav files....
正在从声音解码文件

```

上面程序演示了隐藏信息与解码信息的过程。程序运行后，先将 Python 源代码文件 4-1.

py 隐藏在一段噪声中。然后，从噪声中解码信息，将文本内容输出到 mytext.txt 中，用记事本打开该文件，如图 4-29 所示。可以看到解码后的程序和源程序一样，包括空格、符号及字母等。

```

mytext.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
#!/usr/bin/env python
#4-1.py
import cv2

fn="test1.jpg"

if __name__ == '__main__':
    print 'http://blog.csdn.net/myhaspl'
    print 'myhaspl@qq.com'
    print
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    cv2.imshow('preview', img)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

图 4-29 解码后文件

如图 4-30 所示是程序运行后生成的声音波形图像，很难看出来哪些采样点隐藏了文本信息。下载本书源码包后，可以播放这段声音（运行 4-18.py 后，会产生声音文件 pltest.wav），只能听出声音是一段很平常的噪声。

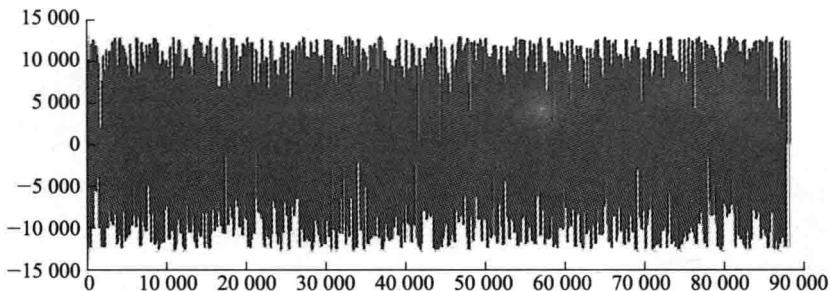


图 4-30 隐藏了文本信息的波形

3. 信息隐藏与解码总结

综上所述，隐藏信息的算法过程为：

(1) 读取程序文本文件，将字符转换为 ASCII 码。

(2) 确定声音文件的相关参数，生成 2 秒声音，其中采样数据用随机数代替，生成随机数的取值范围为 -15000~15000，在某些采样点上用来自隐藏信息的字符字节代替，代替的

方式是将字节对应的 ASCII 码乘上某个基数加一个调整参数，代替的过程是线性的。

(3) 将生成的声音数据写入载体文件中。

解码信息的算法过程为：

(1) 读取载体声音文件及其相关参数。

(2) 按照隐藏信息时的规律，在正确的位置读取字符，然后将读取的字符合成信息。

(3) 将信息写入恢复文件中。

4.2 R 语言基础

R 是用于统计分析、绘图的语言和操作环境，是统计计算和统计制图的优秀工具，属于 GNU 系统的一个自由、免费、源代码开放的软件。其 GUI 界面主要包括菜单、命令控制台，在 Windows 平台下的界面如图 4-31 所示。

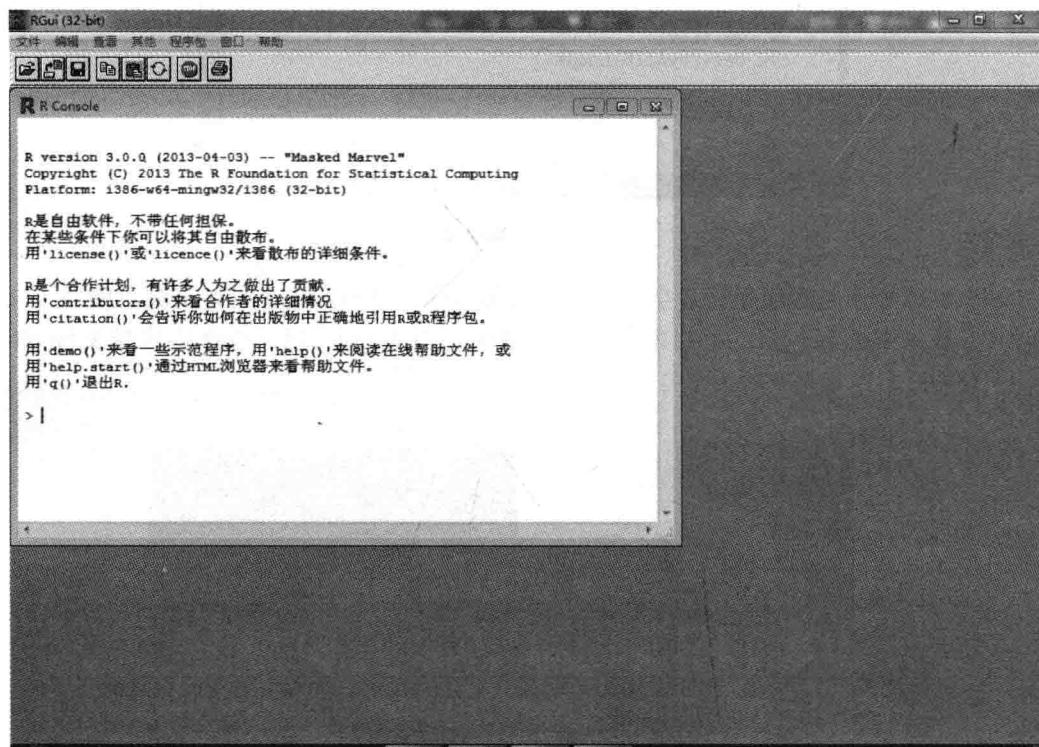


图 4-31 R 的 GUI 界面

4.2.1 基本操作

1. 提示符

R 以“>”为 shell 提示符，与 Windows、Linux、MAC 一致。

2. 获得帮助的方式

在 R 中使用 help 函数获取某个命令或函数的帮助。下面是获取求平均值函数的帮助函数：

```
> help(mean)
starting httpd help server ... done>
```

输入上述命令后，显示如下 HTML 形式的帮助文档：

```
mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

如果想进一步获得这个函数的调用示例，可以通过 example 命令。

```
> example(mean)
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

3. 文件载入并执行代码

使用 source 函数载入并执行代码，把以下代码放在一个名为 test.r 的文件，用文本编辑工具录入。

```
x<-c(22,23,44,66);
y<-mean(x);y
```

然后加载执行，查看输出结果。

```
> source("f:/pro/r/test.r")
> y
[1] 38.75
> x
[1] 22 23 44 66
```

最后将执行结果写入文件。

```
> sink("f:/pro/r/test.lis")
> x
> y
```

打开 test.lis，可看到以下内容：

```
[1] 22 23 44 66
[1] 38.75
```

如果采用不带参数的 sink，将恢复结果。示例如下：

```
> sink()
> x
[1] 22 23 44 66
```

4. 代码续行

在行尾使用“+”可进行续行。

```
x<-c(11,22,33+
+ 22+
+ 333)
> x[1] 11 22 3388
```

另外，需要提一下，R语言中的注释可以被放在任何地方，只要是以井号（#）开始，到行末结束就可以。

5. 物件（对象集）

在R中创建的单元为物件（对象集），这些物件可以是变量、数字数组、字符串、函数，以及从这些物件中产生的更多结构。

objects() 可用来显示存储在 R 中的对象集名字。

```
> objects()
[1] "x"   "xm"
```

以上代码显示 R 目前运行环境中有 x 和 xm 两个变量，可以使用 rm 移除某个对象。

```
> rm(xm)
> objects()
[1] "x"
```

退出 R 程序时，可以以 .RData 的方式保存这些对象集，如图 4-32 所示。



图 4-32 保存对象集对话框

当下次再启动 R 时，这个对象集会被还原。

```
R version 3.0.0 (2013-04-03) -- "Masked Marvel"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)
```

.....

原来保存的工作空间已还原。

```
> x
[1] 11 22 3388
```

4.2.2 向量

1. 数据型向量及其运算

最简单的数据结构是数字型向量，它是一个有序的数字集合（这里的序不是指按数字大小排序，是指数字之前的先后顺序都确定），关于这个序，数学上有一个很好的解释，叫作偏序关系。

偏序关系又称半序关系。设 A 是一个非空集， P 是 A 上的一个关系， P 适合下列条件：

- (1) 对任意的 $a \in A, (a,a) \in P$ ；
- (2) 若 $(a,b) \in P$ 且 $(b,c) \in P$ ，则 $a=b$ ；
- (3) 若 $(a,b) \in P, (b,c) \in P$ ，则 $(a,c) \in P$ ，则称 P 是 A 上的一个偏序关系。带偏序关系的集合 A 称为偏序集或半序集。

比如说 (1,2) 和 (2,1)，它们两个就不是同个向量，因为这两个集合的序不一样。

向量的使用方法很简单，可使用 "c" 后跟括号将向量包围起来，即 c() 函数。如：

```
> y<-c(12,33,12,22)
> y
[1] 12 33 12 22
```

"<-" 可以相当于 "="，就是将这个向量组作为 y 这个对象的值，也可以使用 assign() 函数。

```
> assign("x",c(11,22,15))
> x
[1] 11 22 15
```

"->" 的作用与 "<-" 类似。

```
> c(12,33,12,22)->z
> z
[1] 12 33 12 22
```

对向量操作，一般是对向量的每个元素进行操作，比如：

```
> z
[1] 12 33 12 22
> z/2
[1] 6.0 16.5 6.0 11.0
```

向量也可以成为 c() 中的参数，向量中的元素，将合并成为 C() 函数中的元素：

```
> c(33,12,66)->x1
> y1=c(x1,111,x1)
```

```
> y1
[1] 33 12 66 111 33 12 66
```

再来看看数字型向量运算。向量之间的运算是每个元素分别进行的，比如：

```
> x
[1] 11 22 3388
> 2*x->y
> y
[1] 22 44 6776
> x+3*y->z
> z
[1] 77 154 23716
```

元素个数不一致的向量，元素个数较少的向量将循环扩充和元素个数最多的向量一致，这意味着元素数量最多的向量的元素个数必须是元素数量小的向量的元素个数的整数倍。

```
> z
[1] 77 154 23716

> bb<-c(12,21,32,60,132,56)
> z/3+bb
[1] 37.66667 72.33333 7937.33333 85.66667 183.33333 7961.33333
```

对向量元素的操作，可以使用普通的 +、-、*、/、^ 等操作符，也可以使用更多的函数，比如：log、sin、tan、max、mean、sum 等，这些函数有些是对每个元素分别计算，有些是对所有元素一起计算。

```
> x
[1] 11 22 3388
> cos(x)#cos三角函数
[1] 0.004425698 -0.999960826 0.206187272
> sin(x)#sin三角函数
[1] -0.999990207 -0.008851309 0.978512549
> sum(x) #求和
[1] 3421
> mean(x) #求平均值
[1] 1140.333
```

可以使用 sort、length、sqrt 对向量进行排序，求长度，求平方根。

```
> c(4,8,9)->x
> sqrt(x) #平方根
[1] 2.000000 2.828427 3.000000
> length(x) #长度
[1] 3
> sort(x)->y#排序
> y
[1] 4 8 9 12
```

2. 复数向量

复数向量的元素都是复数。复数的表示方法是：实部 + 虚部 i。下面的代码演示了复数

向量的使用方法。

```
>c(2+1i,3-9i,4,6+1i)->b
>y
[1] 4 8 9 12
>b+y->w#复数运算
>w
[1] 6+1i 11-9i 13+0i 18+1i
```

我们可以使用 `1:m-1` 和 `1:(m-1)` 产生规则的序列。

```
>c(1:(22))
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
>c(1:22)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
```

冒号的优先权很高，看下面这个示例，它产生范围在 3-30 之内的公差为 3 的等差数列。

```
>c(3*(1:10))
[1] 3 6 9 12 15 18 21 24 27 30
>c(3*1:10)
[1] 3 6 9 12 15 18 21 24 27 30
```

`seq` 函数是生成序列的最好工具。可以使用它产生符合某种规则的序列，`seq` 函数有 5 个参数，前 4 个参数分别是起始值（参数名称：`from`）、终止值（参数名称：`to`）、步长（参数名称：`by`）、长度即元素个数（参数名称：`length.out`）。

```
>seq(1,5)
[1] 1 2 3 4 5
>seq(1,5,2)
[1] 1 3 5
```

可以直接指定参数名称，传入参数。

```
>seq(from=1,to=15,by=2)
[1] 1 3 5 7 9 11 13 15
>
>seq(from=1,to=15,by=3)
[1] 1 4 7 10 13
>seq(from=1,to=15,length.out=3)
[1] 1 8 15
```

第五个参数是 `along.with`，使用 `along.with` 参数中序列的长度作为要产生序列的长度。

```
>seq(from=2,along.with=c(1:5))
[1] 2 3 4 5 6
>seq(from=2,by=2,along.with=c(1,2,5,8))
[1] 2 4 6 8
>seq(from=2,by=2,along.with=c(1,2,3,8))
[1] 2 4 6 8
```



`along.with` 参数中的序列仅取其长度，和序列的内容无关。

`rep` 函数对序列中的元素进行重复后拼接，拼接的方式是：使用 `times` 参数将所有元素作为整体拼接，使用 `each` 参数将元素分别进行拼接。

```
> rep(x,2)
[1] 12 32 98 12 32 98
> rep(x,3)
[1] 12 32 98 12 32 98 12 32 98
> rep(x,times=2)
[1] 12 32 98 12 32 98
> > rep(x,each=2)
[1] 12 12 32 32 98 98
```

3. 逻辑型向量

了解了数字型向量，再来看看逻辑型向量。该向量的元素由逻辑型值组成，逻辑型的值有 `TRUE`（可缩写成 `T`）、`FALSE`（可缩写成 `F`）、`NA`（即无效）等，可使用 `>`、`>=`、`==`、`!=` 等逻辑操作符，`and` 操作用 `&`，`or` 操作用 `|`，逻辑非使用 “`!`”。

```
C(12,33,51)->x
> x
[1] 12 33 51
> x>20->y
> y
[1] FALSE TRUE TRUE
> x>=12->y#x>12作为产生逻辑向量的依据
> y
[1] TRUE TRUE TRUE
> x>=12&x<30->y
> y
[1] TRUE FALSE FALSE
> x>=12|x<30->y
> y
[1] TRUE TRUE TRUE
>> !(x>=12&x<30)->y
> y
[1] FALSE TRUE TRUE
```

无效值或缺失值 `NA`、`NaN` 主要用于应付某操作没完成，结果未知的情况。示例如下：

```
> c(1:4,NA,2:3)->x
> x
[1] 1 2 3 4 NA 2 3
> is.na(x)
[1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

下面是数字计算对 `NAN` 的处理：

```
> 0/0
[1] NaN
> 0/0->y
> is.na(y)
[1] TRUE
> y
```

```
[1] NaN
```

4. 字符串向量

字符串用单引号或双引号包围，示例如下：

```
> c("qq", "bb") ->z
> z
[1] "qq" "bb"
```

可在字符串中使用转义符\：

```
\n 新行
\r 回车
\t tab
\b 退格
\a 鸣叫
\\ \
\' ,
\" "
```

在字符串向量的运算中，paste() 函数接受任意数量的参数，可将它们依次连接到字符串向量的元素中，sep 指定连接时相隔的字符，默认为单个空格。

```
> paste(1:12)
[1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
> paste("A", 1:6, sep = "")
[1] "A1" "A2" "A3" "A4" "A5" "A6"
> paste("A", 1:6)
[1] "A 1" "A 2" "A 3" "A 4" "A 5" "A 6"
> paste("A", 1:6, sep = "")
[1] "A1" "A2" "A3" "A4" "A5" "A6"
> paste(c("A", "B"), 1:10, sep="")
[1] "A1"  "B2"  "A3"  "B4"  "A5"  "B6"  "A7"  "B8"  "A9"  "B10"
> paste("今天是", date())
[1] "今天是 Sun Apr 21 14:18:38 2013"
>
```

5. 索引向量

在逻辑值型索引中，索引向量的元素为逻辑值型，逻辑值为 TRUE 的向量将被放在输出结果中。示例如下：

```
> x
[1] 11 22 3388
> x>22->jg
> jg
[1] FALSE FALSE TRUE
> x[jg]->y
> y
[1] 3388
> x[x<100]->y
> y
[1] 11 22
```

>

在正整数型索引中，索引向量是正整数类型，用于指示要哪些位置的元素要输出到结果中。示例如下：

```
> x[c(1,2,3,2,1)]#以向量作为索引
[1] 11 22 3388 22 11
> x
[1] 11 22 3388
> x[1]
[1] 11
> x[1:2]
[1] 11 22
> x[2:3]
[1] 22 3388
```

对于负数索引，会将除开索引以外的所有元素输出到结果中。示例如下：

```
> c(12,22,88)->x
> x
[1] 12 22 88
> x
[1] 12 22 88
> x[-(1:2)]->y
> y
[1] 88
> x[-(1)]->y
> y
[1] 22 88
```

在字符串索引中，是以字符串来标注元素的位置的。示例如下：

```
> c(23,26,27)->age
> c("张三","李四","王五")->names(age)
> age[c("张三")]#以字符串作为索引
张三
23
> age[c("张三","王五")]>mystudent
> mystudent
张三 王五
23 27
```

带索引方式的向量对象可以直接作为被赋值的对象，所有在索引内的向量元素都会被赋值。示例如下：

```
> age[c("张三","王五")]
张三 王五
26 28
> age[c("张三","王五")]<-age[c("张三","王五")]+1
> age[c("张三","王五")]
张三 王五
27 29
> age[c("张三","王五")]<-32
```

```
> age[c("张三", "王五")]
张三 王五
32    32
```

再举个例子：给所有小于 100 的元素均加上 20，代码如下：

```
> x
[1] 11 22 3388
> length(x)
[1] 3
> x[x<100]<-x[x<100]+20
> x
[1] 31 42 3388
```

4.2.3 对象集属性

1. 固有属性

对象集的固有属性有 mode 和 length 两种，相关示例如下：

```
> x
[1] 11 22 3388
> mode(x)
[1] "numeric"
```

其中 mode 可理解为对象集的类型，主要有 numeric、complex、logical、character 和 raw 等类型。Mode 的用法如下：

```
> length(x)
[1] 3
> c(1.0-5i, 20+51i)->a
> mode(a)
[1] "complex"
```

mode 属性转化可完成数据类型的转化。比如，使用 as.character 转化为字符型等。示例如下：

```
> h<- 5:12
> h
[1] 5 6 7 8 9 10 11 12
> as.character(h) #转化为字符
[1] "5" "6" "7" "8" "9" "10" "11" "12"
> c(1.0-5i, 20+51i)->a
> mode(a)
[1] "complex"
> as.character(a)->c_a
> c_a
[1] "1-5i" "20+51i"
```

2. 设置对象属性

可使用 attr 方法进行属性的自定义。具体方法为：使用 attr(object, name) 格式设置对象属性。示例如下：

```

> h
[1] 5 6 7 8 9 10 11 12
> attr(h,"name")
[1] "test"
> attr(h,"name")
[1] "test"
> h
[,1] [,2] [,3] [,4]
[1,] 5 7 9 11
[2,] 6 8 10 12
> attr(,"name")
[1] "test"

```

4.2.4 因子和有序因子

因子用来存储类别变量和有序变量，可用来分组或分类，因子表示分类变量，有序因子表示有序变量。

在 R 语言中使用 `factor()` 函数生成因子对象，语法是 `factor(data, levels, labels, ...)`，其中 `data` 是数据，`levels` 是因子水平向量，`labels` 是因子的标签向量。

```

> my_num<-c(11,22,34,71,14,68,21)
> factor(my_num)->nums#生成my_num分组向量

```

生成因子对象后，输入对象名称，可显示简单的分类情况。

```

> nums
[1] 11 22 34 71 14 68 21
Levels: 11 14 21 22 34 68 71
>

```

`Levels` 函数用于生成因子向量中的水平（去除重复元素后的元素集）。示例如下：

```

> my_num<-c(11,22,34,71,14,68,21,22,11,34)
> factor(my_num)->nums
> nums
[1] 11 22 34 71 14 68 21 22 11 34
Levels: 11 14 21 22 34 68 71
> levels(nums)
[1] "11" "14" "21" "22" "34" "68" "71"

```

还可使用 `ordered` 函数生成有序因子。示例如下：

```

> ordered(nums)
[1] 11 22 34 71 14 68 21 22 11 34
Levels: 11 < 14 < 21 < 22 < 34 < 68 < 71
> age <- c(25, 12, 15, 12, 25)
> ordered(age, levels = c(25,12,15))
[1] 25 12 15 12 25
Levels: 25 < 12 < 15

```

`cut()` 函数将数据转换成因子或有序因子，并进行分组。下面对一组学生成绩进行分组。

```
> score<-c( 88, 85, 75, 97, 92, 77, 74, 70, 63, 97)
```

```

> cut(score, breaks = 3) #将成绩分为3组
[1] (85.7,97]   (74.3,85.7] (74.3,85.7] (85.7,97]   (85.7,97]   (74.3,85.7]
[7] (63,74.3]   (63,74.3]   (63,74.3]   (85.7,97]
Levels: (63,74.3] (74.3,85.7] (85.7,97]
> cut(score, breaks = 2) #将成绩分为2组
[1] (80,97] (80,97] (63,80] (80,97] (80,97] (63,80] (63,80] (63,80]
[10] (80,97]
Levels: (63,80] (80,97]
>

```

4.2.5 循环语句

1. for 循环

R 语言的 for 循环与 Python 类似，都是通过在对象中进行迭代实现循环，但 R 语言中不能在该循环中直接设置起始值、终止值与步长。示例如下：

```

> z<-c()
> x<-(1:10)
> y<-(11:20)
> for (i in 1:length(x)) {
+ z[i]=x[i]^2+y[i]^2
+ }
> z
[1] 122 148 178 212 250 292 338 388 442 500

```

2. while 循环

while 语句每次会检查循环条件，如果条件不再满足，则终止循环。示例如下：

```

> x<-c(1:10)
> i=1
> while (x[i]^2<10) {
+ i=i+1
+ x[i]=x[i]^2
+ }
> x
[1] 1 4 3 4 5 6 7 8 9 10

```

4.2.6 条件语句

if...else... 是 R 语言的条件语句。该语句通过检查执行条件来确定是否继续往下执行，如果条件满足，则执行 if 后面的对应语句，否则执行 else 后面的对应语句。示例如下：

```

> z<-c()
> x<-(1:10)
> y<-(11:20)
> for (i in 1:length(x)) {
+ if ((x[i]^3>y[i]^2))
+ z[i]=x[i]^3
+ }

```

```

+ else
+ z[i]=y[i]^2
+
> z
[1] 121 144 169 196 225 256 343 512 729 1000

```

4.3 R 语言科学计算

4.3.1 分类(组)统计

R 语言分类统计主要包括数据分类整理、统计函数定义、数据分类统计 3 个过程。下面以对水果的价格进行分类统计为例说明。

1. 准备分组数据

```

> fruit_class<-c("苹果","梨子","橘子","草莓","苹果","橘子","橘子","草莓","橘子","草莓")
> fruit_prices<-c(3.5,2.5,1.5,5.5,4.2,3.2,2.8,4.8,2.9,5.8)

```

2. 平均价格统计

通过在 tapply 函数中指定 mean 函数为其参数，可实现分组求平均值。计算结果分 2 行，第 1 行为组名，第 2 行为 tapply 函数最后一个函数参数的运算结果。示例如下：

```

> tapply(fruit_prices,fruit_class,mean)
    草莓    橘子    梨子    苹果
5.366667 2.600000 2.500000 3.850000

```

3. 最低价格统计

通过在 tapply 函数中指定 min 函数为其参数，可实现分组求平均值。示例如下：

```

> tapply(fruit_prices,fruit_class,min)
    草莓    橘子    梨子    苹果
4.8 1.5 2.5 3.5

```

4. 标准差统计

通过在 tapply 函数中指定 sd 函数为其参数，可实现分组求标准差。示例如下：

```

> tapply(fruit_prices,fruit_class,sd)
    草莓    橘子    梨子    苹果
0.5131601 0.7527727 NA 0.4949747

```

5. 标准误

标准误即样本均数的标准差，是描述均数抽样分布的离散程度及衡量均数抽样误差大小的尺度，反映的是样本均数之间的变异。

但是，请注意，标准误并不是标准差，而是样本均值的标准差，是用来衡量抽样误差的，其计算公式为：

$$S_{\bar{x}} = \frac{S}{\sqrt{n}}$$

其中， S 为样本标准差。标准误越小，表明样本统计量与总体参数的值越接近，样本对总体越有代表性，用样本统计量推断总体参数的可靠程度越大。

可通过在 tapply 函数中指定自定义函数 stderr 为其参数，来实现分组求标准误。示例如下：

```
> stderr <- function(x) sqrt(var(x)/length(x)) # 自定义 stderr 函数
> tapply(fruit_prices, fruit_class, stderr)
    草莓      橘子      梨子      苹果
0.2962731  0.3763863          NA  0.3500000
```

4.3.2 数组与矩阵基础

R 提供了简单的工具以处理数组和矩阵。

1. 数组与矩阵的维数

数组与矩阵的维数是其行向量（或列向量）生成的向量空间的维数，可用维数向量表示，格式为（行数 \times 列数），元素都非负。通常使用 dim 函数来定义数组维度。示例如下：

```
> dim(my_num)<-c(2,5) ### 指定数组维度
> my_num
 [,1] [,2] [,3] [,4] [,5]
[1,]   11   34   14   21   11
[2,]   22   71   68   22   34
> dim(my_num)<-c(10)
> my_num
[1] 11 22 34 71 14 68 21 22 11 34
```

2. 切片

切片是操作多维数据（矩阵、数组等）的主要手段，它以索引为参数获取数组或矩阵的一部分。比如：想要得到多维数组的一个切片，则以索引为下标进行访问，取得某块数组。使用 [索引] 格式的参数对数组和矩阵完成切片操作。

索引的形式主要有以下几种：

(1) [index1, index2, ..., indexn]。index1、index2 等分别标明了元素在相应维数的索引，将索引组合成完整的位置，标注需要取出的元素，进行切片，形成数组块。比如：数组 a 的大小为 3×5 ，a[2,4] 表示第 2 行第 4 列的元素。

(2) [c(index1, index2, ..., indexn)]。index1、index2 等 n 个整数标注了元素的位置，将这些标注的元素取出后，组成数组块。这些元素的位置以列为顺序排列，比如，数组 a 的大小为 3×2 (3 行 2 列)，切片操作 a[c(1,2,3,4,5,6)] 依次取出以下元素：a[1,1]、a[2,1]，a[3,1]、a[1,2]、a[2,2]，a[3,2]。

切片操作示例如下：

```
> h
 [,1] [,2] [,3]
[1,]   12   15   982
[2,]   32   67   321
> c(h[1,2],h[2,3]) ## 获取第1行第2列数据（为15）、第2行第3列的数据（为321）
```

```
[1] 15 321
> h[2,]
[1] 32 67 321
> h[c(1,2,3)]###逐列获取第1、2、3个数据
[1] 12 32 15
> h[6]###逐列获取第6个数据
[1] 321
> h[4]
[1] 67
```

3. 索引向量

数组可作为索引使用，且数组也是向量，因此作为索引的数组可称为索引向量。下面的代码演示了向量 c(1:3,5:4,3:5) 作为索引的情况。

(1) 创建数组 x 和 i, x 是被切片的数组，i 是索引向量。

```
> array(10:20, dim=c(2,5))>x
> x
[,1] [,2] [,3] [,4] [,5]
[1,] 10 12 14 16 18
[2,] 11 13 15 17 19
> array(c(1:3,5:4,3:5), dim=c(2,3))>i
> i
[,1] [,2] [,3]
[1,] 1 3 4
[2,] 2 5 3
```

(2) 以 i 为索引向量提取数组块，索引向量每个元素代表切片的位置，将这些位置指向的元素提取出来，形成数组块（数组切片）。示例如下：

```
> x[i]
[1] 10 11 12 14 13 12
```

(3) 通过索引对数组某个元素赋值。示例如下：

```
> x[i]<-111
> x
[,1] [,2] [,3] [,4] [,5]
[1,] 111 111 111 16 18
[2,] 111 111 15 17 19
```

4. array 函数

array 函数根据维数参数生成多维数组，它的参数主要有两个，第一个是需要形成数组元素的数据，第二个是 dim 参数提示维度。下面的代码演示了 array 函数创建数组的方法，并通过 dim 函数获取数组的大小。

```
> c(1:20)>h
> mya<-array(h, dim=c(4,5))
> mya
[,1] [,2] [,3] [,4] [,5]
[1,] 1 5 9 13 17
[2,] 2 6 10 14 18
```

```
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> mydim<-c(2,10)
> mya<-array(h,dim=c(2,10))
> mya
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17   19
[2,]    2    4    6    8   10   12   14   16   18   20
>
> dim(mya)
[1] 2 10
```

array 函数的第一个参数既可以是向量也可以是单个值，如下所示：

```
> mya<-array(1,dim=c(2,10))
> mya
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    1    1    1    1
[2,]    1    1    1    1    1    1    1    1    1    1
```

5. 数组转换为向量

as.vector 函数可将数组转换为向量。示例如下：

```
> x<-array(c(1:10),dim=c(2,5))
> x
[,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> as.vector(x)
[1] 1 2 3 4 5 6 7 8 9 10
```

6. matrix 矩阵

使用 matrix 函数可创建矩阵（从数学角度定义的矩阵），主要参数为：data 表示构造所需数据，nrow 为行数，ncol 为列数，byrow 表示是否按行顺序分配元素（默认为 FALSE，不按）。

```
> matrix(c(1:10),2,5,TRUE)
[,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
> matrix(c(1:10),2,5)
[,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

7. 对角矩阵

对角矩阵是一个除主对角线上的元素之外皆为 0 的矩阵，对角线上的元素可以为 0 或其他值。通过 diag 函数可生成和分析对角矩阵，如果参数为一维数组，则将参数视为对角线元素，并生成对角矩阵；如果参数为一维以上数组，则将参数视为对角矩阵，对它进行分

析，可提取对角线元素。相关示例如下：

```
> a
[1] 1 2 3 4 5 6 7 8
>
> diag(a)###生成对角矩阵
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    0    0    0    0    0    0    0
[2,]    0    2    0    0    0    0    0    0
[3,]    0    0    3    0    0    0    0    0
[4,]    0    0    0    4    0    0    0    0
[5,]    0    0    0    0    5    0    0    0
[6,]    0    0    0    0    0    6    0    0
[7,]    0    0    0    0    0    0    7    0
[8,]    0    0    0    0    0    0    0    8
> a<-array(c(1:16),dim=c(4,4))
> diag(a)###提取对角线元素
[1] 1 6 11 16
> a
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

4.3.3 数组运算

1. 四则运算

数组四则运算的规律是：对应位置的元素分别计算，而不是依据矩阵的数学运算法则。运算符为“+”（加）、“-”（减）、“*”（乘）等，运算优先级与算术四则运算相同，先乘后加减。示例如下：

```
> mya
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17   19
[2,]    2    4    6    8   10   12   14   16   18   20
> myb
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    2    2    2    2    2    2    2    2    2
[2,]    2    2    2    2    2    2    2    2    2    2
> mya+myb
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    3    5    7    9   11   13   15   17   19   21
[2,]    4    6    8   10   12   14   16   18   20   22
> mya*myb
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    6   10   14   18   22   26   30   34   38
[2,]    4    8   12   16   20   24   28   32   36   40
> 3*mya*myb
```

```

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    6   18   30   42   54   66   78   90  102  114
[2,]   12   24   36   48   60   72   84   96  108  120
>
> mya*myb+mya#先乘后加
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    3     9   15   21   27   33   39   45   51   57
[2,]    6    12   18   24   30   36   42   48   54   60
>

```

2. 向量连接

向量连接指的是将两个向量通过某种规律连接成一个数组。R 语言的 cbind 和 rbind 函数可进行向量连接，其中 cbind 函数将向量的行转变为列后再连接，rbind 函数将向量的列转变为行后再连接。示例如下：

```

> x2<-c(101:105)
> x1<-c(1:10)
> cbind(x1,x2)
  x1  x2
[1,] 1 101
[2,] 2 102
[3,] 3 103
[4,] 4 104
[5,] 5 105
[6,] 6 101
[7,] 7 102
[8,] 8 103
[9,] 9 104
[10,] 10 105
> rbind(x1,x2)
 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
x1    1    2    3    4    5    6    7    8    9    10
x2  101  102  103  104  105  101  102  103  104  105

```

4.3.4 矩阵运算

1. 矩阵连接

R 语言的 cbind 函数完成矩阵的横向连接，rbind 函数完成矩阵的纵向连接。下面是关于矩阵的连接操作示例。

```

> x3<-matrix(c(1:10),2,5)
> x4<-matrix(c(101:105),2,5)
> x3
  [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> x4
  [,1] [,2] [,3] [,4] [,5]

```

```
[1,] 101 103 105 102 104
[2,] 102 104 101 103 105
> cbind(x3,x4)
 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 1 3 5 7 9 101 103 105 102 104
[2,] 2 4 6 8 10 102 104 101 103 105
> rbind(x3,x4)
 [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
[3,] 101 103 105 102 104
[4,] 102 104 101 103 105
```

2. 矩阵转置

线性代数将矩阵 A 的转置（记做 A^T ）定义为：

把 A 的横行写为 A^T 的纵列；

把 A 的纵列写为 A^T 的横行。

根据上述计算法则， $m \times n$ 矩阵 A 的转置生成 $n \times m$ 矩阵 A^T 。

$$A_{ij}^T = A_{ji}, 1 \leq i \leq n, 1 \leq j \leq m.$$

R 语言的 `t` 函数可完成矩阵转置计算。

```
> array(h,dim=c(2,5))->mya
> mya
 [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
> t(mya)###矩阵转置
 [,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
[5,] 9 10
```

相对 `t` 函数而言，用 `aperm` 函数进行矩阵转置更灵活。`aperm` 有两个常用的参数，第一个参数是需要转置的矩阵，第二个参数 `perm` 指示新矩阵相对于第一个参数矩阵的维度下标。需要特别注意的是，第二个参数 `perm` 是维度下标。比如，将行转换为列，将列转换为行，将行列次序更换，将第一维的元素与第二维的元素互换，则将 `perm` 设为 `c(2,1)`。下面的代码演示了 `aperm` 函数的使用方法。

```
> array(h,dim=c(2,5))->mya
> mya
 [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
>
> aperm(mya,perm=c(2,1))->myb##以c(2,1)为维度下标进行转置
> myb
```

```

[,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
>
> array(mya,c(2,2,5))->mya1
> mya1
, , 1
[,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
[,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
[,1] [,2]
[1,]    9    1
[2,]   10    2
, , 4
[,1] [,2]
[1,]    3    5
[2,]    4    6
, , 5
[,1] [,2]
[1,]    7    9
[2,]    8   10
> aperm(myal,perm=c(2,1,3))->myb1###以c(2,1,3)为维度下标进行转置
> myb1
, , 1
[,1] [,2]
[1,]    1    2
[2,]    3    4
, , 2
[,1] [,2]
[1,]    5    6
[2,]    7    8

```

```
, , 3
```

```
[,1] [,2]
[1,]    9   10
[2,]    1    2
```

```
, , 4
```

```
[,1] [,2]
[1,]    3    4
[2,]    5    6
```

```
, , 5
```

```
[,1] [,2]
[1,]    7    8
[2,]    9   10
> aperm(myal,perm=c(1,3,2))->myb1
```

```
> myb1
```

```
, , 1
```

```
[,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9    3    7
[2,]    2    6   10    4    8
```

```
, , 2
```

```
[,1] [,2] [,3] [,4] [,5]
[1,]    3    7    1    5    9
[2,]    4    8    2    6   10
>
```

3. 矩阵乘积

若 A 为 $m \times n$ 矩阵, B 为 $n \times r$ 矩阵, 则它们的乘积 AB (有时记做 $A \cdot B$) 会是一个 $m \times r$ 的矩阵, 前提是 m 与 n 必须相同, 矩阵乘积使用 $\%*\%$ 操作符进行计算。示例如下:

```
> a
[,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> b
[,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
```

```
[5,]    5   10
> a %*% b
[,1] [,2]
[1,]   95  220
[2,]  110  260
```

4. 内积运算

(1) 向量外积。向量的外积是矩阵的克罗内克积的特殊情况。给定 $m \times 1$ 列向量 u 和 $1 \times n$ 行向量 v , 它们的外积 $u \otimes v$ 被定义为 $m \times n$ 矩阵 A 。

$$u \otimes v = A = uv$$

向量外积 $u \otimes v$ 的计算定义为:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \otimes \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_1 & a_3 b_1 \\ a_1 b_2 & a_2 b_2 & a_3 b_2 \\ a_1 b_3 & a_2 b_3 & a_3 b_3 \\ a_1 b_4 & a_2 b_4 & a_3 b_4 \end{bmatrix}$$

下面的代码演示了 a 和 b 数组作为向量的外积运算和普通乘法运算。

```
> b<-array(c(1:4))
> a<-array(c(5:6))
> b%o%a##外积
[,1] [,2]
[1,]    5    6
[2,]   10   12
[3,]   15   18
[4,]   20   24
> b
[1] 1 2 3 4
> a
[1] 5 6
> b<-array(c(1:4))
> a<-array(c(5:8))
> a*b##普通乘法
[1]  5 12 21 32
> b
[1] 1 2 3 4
> a
[1] 5 6 7 8
> a%o%b
[,1] [,2] [,3] [,4]
[1,]    5   10   15   20
[2,]    6   12   18   24
[3,]    7   14   21   28
[4,]    8   16   24   32
>
```

此外, 还可以使用 `Outer(a,b, “*”)` 替代 `%o%` 运算符进行外积运算。

(2) 向量内积。向量内积以实数 \mathbb{R} 上定义的两个向量为运算对象, 返回一个实数标量

值，属于二元运算，它是欧几里得空间的标准内积。

两个向量 $a = [a_1, a_2, \dots, a_n]$ 和 $b = [b_1, b_2, \dots, b_n]$ 的内积定义为：

$$a \cdot b = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

R 语言通过 `crossprod` 函数完成向量内积计算。示例如下：

```
> a<-c(1:3)
> b<-c(4:6)
> crossprod(a,b)
[,1]
[1,] 32
> a<-c(1:3)
```

(3) 矩阵内积。矩阵内积的计算方式相当于第一个参数的转置乘以第二个参数，就是前面提到的矩阵乘积。除了使用 `%*%` 操作符外，还可以使用 `crossprod` 函数完成矩阵内积计算。示例如下：

```
> b<-array(c(4:6),dim=c(1,3))
> a<-array(c(1:3),dim=c(1,3))
> a
[,1] [,2] [,3]
[1,] 1 2 3
> b
[,1] [,2] [,3]
[1,] 4 5 6
> crossprod(a,b)###a与b完成矩阵内积计算
[,1] [,2] [,3]
[1,] 4 5 6
[2,] 8 10 12
[3,] 12 15 18
> t(a) %*% b##a的转置与b完成矩阵内积计算
[,1] [,2] [,3]
[1,] 4 5 6
[2,] 8 10 12
[3,] 12 15 18
```

5. 求解线性方程组

一般通过 `solve` 函数来求解 $a \%*% x = b$ 中的 x 向量值，求解线性方程组仅使用 `solve` 函数的前两个参数，第一个 a 为系数矩阵，第二个 b 为常数项，当 b 缺失时，默认为单位矩阵。示例如下：

```
> b
[,1]
[1,] 8
[2,] 9
> a
[,1] [,2]
[1,] 1 3
```

```
[2,]    2    4
> solve(a,b)
[,1]
[1,] -2.5
[2,]  3.5
```

6. 矩阵求逆

通过 `solve` 函数可进行矩阵求逆计算，只指定 1 个参数（待求逆的矩阵）即可。示例如下：

```
> a
[,1] [,2]
[1,]    1    3
[2,]    2    4

> solve(a)
[,1] [,2]
[1,]   -2   1.5
[2,]    1  -0.5
>
```

7. 矩阵的特征值求解

(1) 特征值概念。 λ 是 A 的特征值等价于线性系统 $(A-\lambda I)v=0$ (其中 I 是单位矩阵) 有非零解 v (一个特征向量)，特征值存在等价于下面行列式成立：

$$\det(A-\lambda I)=0$$

函数 $p_A(\lambda)=\det(A-\lambda I)$ 是一个关于 λ 的多项式，称为 A 的特征多项式。矩阵的特征值也就是其特征多项式的零点。

求一个矩阵 A 的特征值可以通过求解方程 $p_A(\lambda)=0$ 来得到。

(2) R 语言求解特征值。利用 R 语言的 `eigen` 函数可求解特征值，调用格式如下：

```
eigen(x, symmetric, only.values = FALSE)
```

其中， x 为需要求特征值的矩阵；`symmetric` 是逻辑型，表示是否为对称矩阵，对称矩阵是一个方形矩阵，其转置矩阵和自身相等，即： $A=A^T$ ，对称矩阵 $A=(a_{ij})$ 从右上至左下方的元素以主对角线（左上至右下）为轴对称，即： $a_{ij}=a_{ji}$ 。`only.values` 如果为 TRUE，则只返回特征值，否则返回特征值和特征向量。

下面的代码演示了 `eigen` 函数计算特征值的方法。

```
> a<-array(c(1:16),dim=c(4,4))
> eigen(a)##计算a的特征值
$values
[1]  3.620937e+01 -2.209373e+00  1.599839e-15  7.166935e-16

$vectors
[,1]           [,2]           [,3]           [,4]
[1,] 0.4140028  0.82289268 -0.5477226  0.1125155
[2,] 0.4688206  0.42193991  0.7302967  0.2495210
```

```
[3,] 0.5236384 0.02098714 0.1825742 -0.8365883
[4,] 0.5784562 -0.37996563 -0.3651484 0.4745519

>
> eigen(a,only.values=FALSE)
$values
[1] 5.3722813 -0.3722813

$vectors
[,1] [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648 0.4159736
```

8. 求解矩阵行列式

行列式是线性代数中的一个概念，将一个 $n \times n$ 的矩阵 A 映射到一个标量，记作 $\det(A)$ 或 $|A|$ 。行列式可看作是有向面积的概念在一般的欧几里得空间中的推广。

比如，假设矩阵 A 定义为：

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

则矩阵 A 的行列式 $|A|$ 可定义为：

$$|A| = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

R 语言的 `det` 函数可求解矩阵对应的行列式值，即：已知矩阵 A ，求解 $|A|$ 。示例如下：

```
> a<-array(c(1:4),dim=c(2,2))
> det(x)
[1] -2
>
```

9. 奇异分解

奇异值分解是线性代数中一种重要的矩阵分解，在信号处理、统计学等领域有重要应用。假设 M 是一个 $m \times n$ 阶矩阵，其中的元素全部属于域 K （实数域或复数域）。设存在一个分解使得：

$$M = U \Sigma V^*$$

其中 U 是 $m \times m$ 阶酉矩阵； Σ 是半正定 $m \times n$ 阶对角矩阵； V^* （即 V 的共轭转置）是 $n \times n$ 阶酉矩阵。

这种分解称作 M 的奇异值分解， Σ 对角线上的元素 Σ_{ii} 即为 M 的奇异值。

使用 R 语言的 `svd` 函数可完成奇异分解。示例如下：

```
> array(c(1:16),dim=c(4,4))>a
> a
[,1] [,2] [,3] [,4]
```

```
[1,]    1     5     9    13
[2,]    2     6    10    14
[3,]    3     7    11    15
[4,]    4     8    12    16
> svd(a)###奇异分解
$d
[1] 3.862266e+01 2.071323e+00 1.291897e-15 6.318048e-16
$u
[,1]      [,2]      [,3]      [,4]
[1,] -0.4284124 -0.7186535  0.5462756 -0.0397869
[2,] -0.4743725 -0.2738078 -0.6987120  0.4602190
[3,] -0.5203326  0.1710379 -0.2414027 -0.8010772
[4,] -0.5662928  0.6158835  0.3938391  0.3806452
$v
[,1]      [,2]      [,3]      [,4]
[1,] -0.1347221  0.82574206 -0.4654637 -0.2886928
[2,] -0.3407577  0.42881720  0.4054394  0.7318599
[3,] -0.5467933  0.03189234  0.5855124 -0.5976414
[4,] -0.7528288 -0.36503251 -0.5254881  0.1544743
```

4.4 R 语言计算实例

4.4.1 学生数据集读写

下面演示 R 语言对学生数据集的操作。R 语言可以使用 list (列表) 组件创建与读写学生数据，该组件通常用来容纳一个数据集，其中包含不同的数据类型。

(1) 创建学生数据集 (创建列表的语法是：list (字段 1= 组件 1, 字段 2 = 组件 2, ...)), 学生数据集由 3 个不同类型的数据组成：name (姓名，类型为字符型)、class (班级，类型为字符型)、ages (年龄，类型为数值型)。

```
> list(name="students",class="101",stdt.ages=c(22,25,20),stdt.name=c("zhangsang","lisi","wangwu"))->mystudents
```

(2) 读取列表。下面的代码读取刚才创建的学生数据集：

```
> mystudents
$name
[1] "students"

$class
[1] "101"

$stdt.ages
[1] 22 25 20

$stdt.name
[1] "zhangsang" "lisi"      "wangwu"
```

(3) 获取学生数据集的字段总数 (通过 length 返回 list 组件的数量)。

```
> length(mystudents)
[1] 4
```

(4) 查看数据集中所有学生的姓名和年龄(通过“列表变量名\$字段名”提取组件内容)。

```
> c(mystudents$name, mystudents$age)
[1] "zhangsang" "lisi"      "wangwu"    "22"       "25"       "20"
```

此外, R语言还提供了一个很不错的list组件data.frame, 它内部可拥有很多组件。下面接着以学生数据为例讲解data.frame。

(1) 创建data.frame组件, 存储学生数据。

```
>
data.frame(name=mystudents$name, age=mystudents$age) ->mysts
> mysts
  name age
1 zhangsang 22
2      lisi 25
3     wangwu 20
```

(2) 将数据集中的年龄都增长1岁(随着新的一年到来, 学生们都长大了1岁), 完成这个操作可使用attach和detach方法。

前面一直用\$符号访问list列表的字段, 当R语言的代码较多时, 列表组件名前缀访问字段很不方便, 因此, R语言提供了另一对非常有用的工具attach和detach: attach把数据集的所有字段复制一份副本, 绑定在搜索路径, 这样可以直接读取它们(仅能读取, 写回没有意义, 因为这只是副本而已), 无需显示表明列表名字; detach则进行解绑。

(1) 用attach将学生数据集的字段副本绑定在搜索路径中。

```
> attach(mysts)
> age
[1] 22 25 20
> name
[1] zhangsang lisi      wangwu
Levels: lisi wangwu zhangsang
```

(2) 将绑定的age字段副本加1, 并显示更新后的学生数据。

```
> age+1->mysts$age
> mysts
  name age
1 zhangsang 23
2      lisi 26
3     wangwu 21
```

(3) 使用detach将字段副本从搜索路径上删除(解绑)。

```
> detach(mysts)
> age
错误: 找不到对象'age'
> name
错误: 找不到对象'name'
```

4.4.2 最小二乘法拟合

1. 最小二乘法与回归

最小二乘法是一种数学优化技术，它通过最小化误差的平方和找到一组数据的最佳函数匹配。

假设存在 (x, y) 这两个变量，对于一系列的 x 变量值，有一系列的 y 值与其对应，可以找到这两个变量之间的相互关系。比如：对于一次函数来说，可将这些 (x, y) 值标注在直角坐标系统，从而得到一条直线，这些点就在这条直线附近。那么，直线方程的定义为：

$$y = kx + b$$

其中： k 、 b 是任意实数， k 为斜率， b 为截距。

下面以 $y_1=3x+12$ 和 $y_2=6x+12$ 为例进行分析。如图 4-33 所示，实线为 y_1 的图像，虚线为 y_2 的图像。从图像能直观看出，斜率越大，直线越陡。 y_1 的斜率是 3，截距为 12； y_2 的斜率是 6，截距为 12， y_2 方程的图像明显比 y_1 陡。

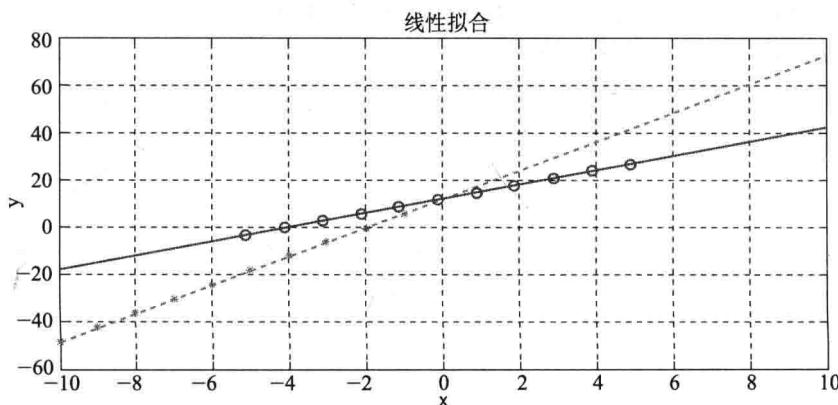


图 4-33 直线的图像

图 4-32 中标注的圆圈和星号为 (x, y) 格式表示的二维数据点，很明显，圆圈的数据点位于 $y_1=3x+12$ 上，而星号的数据点位于 $y_2=6x+12$ 上。这些二维数据点被 $y_1=3x+12$ 和 $y_2=6x+12$ 方程的图像连接。这样做的好处是，我们不需要记忆这些数据点的坐标就能预测类似数据点的位置。比如说，已知 x 为 1.5 时，想要求圆圈数据点的坐标，可直接将 $x=1.5$ 代入 y_1 方程得到：

$$y = 1.5 \times 3 + 12 = 16.5$$

这样就可用 $y = kx + b$ 形式的一次方程拟合数据点，这个过程为线性拟合。拟合的目标是这些点到这条直线的距离的平方和最小。最小二乘法是效果较好的线性拟合方法，最小二乘法拟合数据点的过程就是对数据做回归分析，我们把类似图中的这几条直线称为回归线。

2. 最小二乘法拟合

R 语言提供了 `lsfit` 函数，可完成最小二乘法拟合，其主要参数如下。

- X: 一个矩阵的行对应的情况和其列对应为变量。
- Y: 结果, 可以是一个矩阵。
- Wt: 可选参数, 加权最小二乘法的执行权重向量。
- Intercept: 是否应使用截距项。
- Tolerance: 公差将用于矩阵分解。
- Yname: 用于响应变量的名称。

下面来看看 $y=2x$ 回归方程拟合, 这里以 $x=(1,2,3,4)$, $y=(2,4,6,8)$ 为例在R中进行数据拟合。

```
> y<-c(2,4,6,8)
> x<-c(1,2,3,4)
> lsfit(x,y)###下面的x为常数项, Intercept为截距
$coefficients
Intercept      x
0              2
....
```

上述拟合结果中, Intercept项表示截距, x项表示方程的x变量的常数项, 因此, 回归方程为 $y=2x+0=2x$ 。

再来看看 $y=2x+3$ 回归方程拟合, 设截距为3。修改刚才的方程, 假设回归线为: $y=2x+3$ 。

(1) 根据回归线构造x和y值。

```
> y<-c(5,7,9,11)
> x<-c(1,2,3,4)
```

(2) 执行lsfit()函数进行拟合。

```
> lsfit(x,y)
$coefficients
Intercept      x
3              2
```

上面lsfit()函数的运行结果表明, 这些数据点的回归方程为 $y=2x+3$ 。

4.4.3 交叉因子频率分析

交叉因子频率分析的作用在于分析数据的分布区间及其统计指标。下面举例说明分析过程。

(1) 划分数据分布区间。使用cut函数将变量y中存储的数字划分到5个分布区间: [11, 15]、[15, 19]、[19, 23]、[23, 27]、[27, 31]示例如下:

```
> y<-c(11,22,13,14,11,22,31,31,31,14)
> cut(y,5)->cuty
> cuty
[1] (11,15] (19,23] (11,15] (11,15] (11,15] (19,23] (27,31] (27,31] (27,31]
[10] (11,15]
Levels: (11,15] (15,19] (19,23] (23,27] (27,31]
```

(2) 使用table函数统计数据在每个区间出现的频率。代码如下:

```
> table(cuty)
cuty
[11,15] (15,19] (19,23] (23,27] (27,31]
      5        0        2        0        3
```

(3) 使用 hist 函数生成分布直方图, 以便更直观地观察数据分布情况, 如图 4-33 所示。通过指定 breaks 参数 (设置为各区间的边界值) 和 axes 参数 (设置为 FALSE 表示手动画刻度), 将数据在 table 函数生成的区间内进行划分。代码如下:

```
> bins<-seq(min(y),max(y),by=4)
> hist(y,breaks=bins,col="lightblue",axes=FALSE)
> axis(1,bins)
> axis(2)
```

结合 table 函数的执行结果以及 hist 函数生成的直方图, 可得到以下结论:

(1) 分析 table 函数的执行结果可看出, 数据主要集中在 [11,15] 区间中, [11,15] 区间内分布的数字最多, 该区间内有 5 个数字。此外, 在 [15,19]、[23,27] 区间中没有数据分布, 变量 y 中的数据在这两个区间内出现频率为 0。

(2) 从图 4-34 中可观察到, 数据分布情况与 table 函数执行结果相吻合。

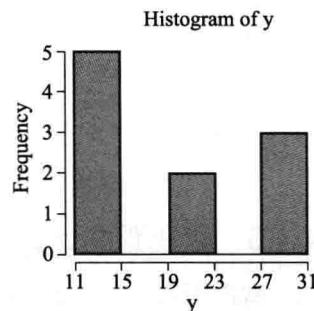


图 4-34 分布直方图

4.4.4 向量模长计算

向量模长即欧几里得范数, 在 n 维欧几里得空间 R_n 上, 向量 $x = (x_1, x_2, \dots, x_n)$ 的长度定义为: $\|x\| := \sqrt{x_1^2 + \dots + x_n^2}$ 。

根据勾股定理, 它给出了从原点到点 x 之间的距离。

1. 模长函数定义

R 拥有自定义函数功能, 可按如下格式定义:

```
函数名<-function(参数1, 参数2, ..., 参数n) {
    函数体
}
```

下面定义求三维向量模长的 vector_length 函数, 完成模长计算。

```
> vector_length<-function(x1,x2,x3) {
+ vlength<-sqrt (x1^2+x2^2+x3^2)
+ vlength
+ }
```

2. 模长计算

调用 vector_length 函数, 计算向量 [12,33,19] 的模长。

```
> vector_length(12,33,19)
[1] 39.92493
```

>

N维向量的模长计算与三维模长类似。下面重新定义 vectorn_length 函数，并调用它计算任意维度向量的模长。

```
> vectorn_length<-function(x) {
+ temp<-0
+ for (i in 1:length(x)) {
+ temp<-temp+x[i]^2
+ }
+ vlength<-sqrt(temp)
+ vlength
+ }
>#下面调用新的vectorn_length函数计算模长
> vectorn_length(c(11,22,33,44,55))
[1] 81.57818
> vectorn_length(c(11,22,33,55))
[1] 68.69498
```

4.4.5 欧氏距离计算

欧氏距离 (Euclid Distance) 是在 n 维空间中两个点之间的真实距离， n 维欧氏空间的每个点可以表示为 $(x[1], x[2], \dots, x[n])$ ， $X = (x[1], x[2], \dots, x[n])$ 和 $Y = (y[1], y[2], \dots, y[n])$ 这两个点之间的距离 $d(X, Y)$ 定义为下面的公式：

$$d(X, Y) = \sqrt{\sum ((x[i]-y[i])^2)} \text{ 其中 } i = 1, 2, \dots, n$$

利用 R 语言的操作符自定义功能完成欧氏距离计算，操作符的定义使用 % 符号 % 的方式定义，实际使用时，% 也属于操作符的一部分。操作符的定义格式如下：

```
操作符名<-function(参数1, 参数2, ..., 参数n) {
    语句
}
```

下面定义 %~% 操作符，并计算二维空间的欧氏距离。示例如下：

```
> "%~%"<-function(x1,x2) {
+ temp<-0
+ for (i in 1:length(x1)) {
+ temp<-temp+(x1[i]-x2[i])^2
+ }
+ edis<-sqrt(temp)
+ edis
+ }
>#使用%~%操作符，计算二维空间的欧氏距离
> c(1,2,3) %~% c(5,6,7)
[1] 6.928203
>
```

可以将计算扩展到 n 维空间中。使用 R 语言的不定数量的函数参数机制，来定义 n 维空间的欧氏距离函数 mycount。示例如下：

```

> mycount<-function(...){
+   temp=0
+   for (i in c(...)){
+     temp=temp+1
+   }
+   temp
+ }
> mycount(11,22,33)
[1] 3
> mycount(11,22,33,66)
[1] 4
> mycount(11,22,66)
[1] 3

```

4.5 小结

本章对 Python 语言和 R 语言的语法基础以及相关计算平台 API 进行了讲述，同时，用大量实例讲解了相关计算平台的实际操作。

Python 语言和 R 语言是本书讲解机器学习用到的主要语言，也是机器学习工程应用中可能用到的编程语言，在此建议大家平时多阅读 R 语言和 Python 语言的官网教程和相关计算平台资料，加深对它们的理解。

因本书篇幅有限，更多的关于 R 语言和 Python 语言的资料可以查询相关官网。下面列举了常用的官网链接。

R 语言文档资料链接：

<http://cran.r-project.org/manuals.html>

Python 科学计算库文档资料链接：

<http://docs.python.org/2/>

<http://docs.scipy.org/doc/>

<http://docs.opencv.org/master/modules/refman.html>

从下章开始，我们将正式进入机器学习和统计分析的实战。建议大家在浏览器中将上面的文档链接收藏，以便更好地理解本书内容。此外，从本章开始，每章小结后均有思考题，希望大家在阅读本书的过程中，多动手，多上机操作，理论联系实践才是王道。

思考题

1. 本章中提到了 Python 将信息隐藏在声音和图像载体文件的方法，能不能将隐藏了信息的图像载体文件隐藏在一段音乐之中？



提示：

可以考虑先将隐藏了信息的图像矩阵读入，然后将其作为原始数据混进 WAV 格式的音

乐文件之中，再用原始音乐文件作为解码的密钥，解码图像数据后，用本章介绍的方法从图像数据中恢复文字信息。

2. 用 R 语言分析一个数据集的数据，将数据分为适当的区间，然后统计数据在每个区间的分布数量，并作出直方图。



提示

用因子频率分析方法实现。

3. 用 R 语言在 x 和 y 之间建立回归模型，得出回归直线方程， $x=[1,3,8,9]$, $y=[2,8,23,80]$



提示

使用 R 语言的回归计算函数分析。

第三部分

统计分析实战篇

在终极的分析中，一切知识都是历史；在抽象的意义下，一切科学都是数学；在理性的基础上，所有的判断都是统计。

——C.R.劳

第5章

统计分析基础

统计学用于研究对象的数量性、总体性、变异性，具体说来，就是通过各种统计指标和指标体系来反映对象总体的规模、水平、速度、比例、效益、差异和趋势等。海量数据分析需要一个科学的理论基础，统计学是理论基础之一，统计分析方法也是主要的数据分析方法。

机器学习涉及许多统计分析理论。机器学习应用的统计分析强调实际应用效果，检测损失函数即描述预测与实际之间的偏差；数据分析领域也以统计学为基础，统计学善于对数据建立模型，并对模型做出假设。由此可见，无论是在机器学习方面还是数据分析领域，统计分析理论都有着举足轻重的地位。

本章将以 R 语言为统计分析计算工具讲述统计分析基础，在此之前，请各位读者按照第一部分准备篇中的指导，将 R 语言计算平台搭建好。

5.1 数据分析概述

随着数据分析技术的不断发展，数据分析的定义众说纷纭。作者尝试将数据分析定义为：“数据分析是将数据转化成知识，对数据加以详细的研究和概括总结的过程，它用适当的统计方法对收集来的大量数据进行分析，发现数据的价值，挖掘数据的表征的知识。”

在第二次世界大战中，数据分析第一次登上世界舞台。在 Budiansky、Stephen 所著的《Blackett's War》中，讲述了一个由数学家、物理学家组成的团体，通过对数据进行收集和分析，使用数据引导战争的故事。该团队运用数据分析技术对付纳粹潜艇舰队，并把原始雷达系统收集的数据与实际战争结合起来分析，从而使海岸司令部的飞机拥有恰当的飞行路线，实现最高效的监视。同时他们还证明了一个论断：在一个 15~24 艘船的舰队中，每艘船有 2.3% 被击沉的概率；而在舰队船数高于 45 时，被击沉的概率只有 1.1%。

随着社会的发展，数据库已经深入到各行各业，成为了社会信息记录的重要载体。为了解决数据的查询问题，在数据库发展的早期，SQL 语言诞生并迅速发展，目前它仍然活跃

在数据库领域。近年来，“大数据”一词被炒得很热，大数据已被用于承载数据相关的绝大部分概念，包括：数据海洋、社交媒体分析、下一代数据管理能力、实时数据等。各大公司已经开始理解并实践探索如何处理并分析大量信息。

无限风光在险峰，只有登上山顶，才能看到真正的风光，既然如此，那我们就开始攀登数据分析这座“大山”吧！

5.2 数学基础

统计分析建立在概率与统计的数学基础之上，在此，先简要介绍一下相关数学公式。

1. 概率

概率是数学概率论的基本概念。设随机事件的样本空间为 Ω ，对于 Ω 中的每一个事件 A ，都有实函数 $P(A)$ ，满足：

非负性： $P(A) \geq 0$

规范性： $P(\Omega) = 1$

可加性：对 n 个两两互斥事件 A_1, \dots, A_n 有： $\sum_{i=1}^n P(A_i) = P\left(\bigcup_{i=1}^n A_i\right)$

任意一个满足上述条件的函数 P 都可以作为样本空间 Ω 的概率函数，称函数值 $P(A)$ 为 Ω 中事件 A 的概率。

上面的定义严谨但不易理解，通俗来说，概率是一个 0 到 1 之间的实数，是对随机事件发生的可能性的度量。人们有时候会问：“明天会更冷吗？”、“今天的比赛我们会赢得冠军吗？”等，他们是在关注“天气变冷”、“比赛取胜”等事件发生的机会。在数学上，这些事件发生的机会可用一个数来表示，称为概率。概率的主要公式如下。

设事件 A 发生的概率（可能性）为 $P(A)$ ，它不发生的概率（可能性）为 $P(\bar{A})$ ，它们之间的关系为：

$$P(A) = 1 - P(\bar{A})$$

n 个事件 A_1, \dots, A_n 发生的概率为：

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n P(A_i) - \sum_{1 \leq i \leq j \leq n} P(A_i A_j) + \sum_{1 \leq i \leq j \leq k \leq n} P(A_i A_j A_k) + \dots + (-1)^{n-1} P(A_1 A_2 \dots A_n)$$

条件概率是事件 A 在另外一个事件 B 已经发生条件下的发生概率。条件概率记为 $P(A|B)$ ，读作“在 B 条件下 A 的概率”。比如，“如果明天更冷，需要多穿一件外套吗？”可用条件概率描述为：假设明天天气更冷为事件 B ，多穿一件外套为事件 A ，在事件 B （天气更冷）发生的情况下，事件 A （多穿外套）发生的概率为多少。下面是条件概率的数学定义。

在同一个样本空间 Ω 中的事件或者子集 A 与 B ，如果随机从 Ω 中选出的一个元素属于 B ，那么这个随机选择的元素还属于 A 的概率就定义为在 B 的前提下 A 的条件概率。定义为：

$$P(A|B) = |A \cap B|/|B|$$

再将上式中的分子、分母都除以 $|\Omega|$ ，得到

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

其中， $P(A \cap B)$ 表示联合概率，指 A 和 B 两个事件共同发生的概率，也可以记为 $P(A, B)$ 或 $P(AB)$ 。

条件概率在贝叶斯统计中也称为后验概率，即在考虑和给出相关证据或数据后所得到的条件概率。

现实生活中，使某事件发生的前提（条件概率中的“条件”）可能不止一个。比如：在事件 B_1, \dots, B_n 发生的前提下，事件 A 发生的概率是多少？将其定义如下：

$$P(A) = \sum_{i=1}^n P(AB_i) = \sum_{i=1}^n P(B_i) \cdot P(A|B_i)$$

前面几个公式都是根据前提来推测事件发生的概率的，能不能根据事件发生的概率推测其前提出现的概率呢？当然可以，实质上，这就是根据结论推测原因发生的概率。比如，为什么前面堵车了（事件 A ），是因为发生了交通事故（事件 B_1 ）还是路上车辆太多以致拥堵（事件 B_2 ），或是下完大雨道路状况很差（事件 B_3 ）？

下面的公式定义了事件 A （结论）发生了，事件 B_k （原因）发生的概率，

$$P(B_k|A) = \frac{P(AB_k)}{P(A)} = \frac{P(B_k)P(A|B_k)}{\sum_{i=1}^n P(B_i)P(A|B_i)}$$

2. 概率分布

概率分布简称分布，是数学概率论的一个概念，广义上指随机变量的概率性质，狭义上是指随机变量的概率分布函数，使用最为普遍的是狭义上的定义。在此只讲解狭义概率分布。

(1) 随机变量分布。随机变量的实质是函数，其数学定义为：设函数 X 对于概率空间 S 中每一个事件 e 都有定义 $X(e)$ ，其中函数值为实数，同时，每一个实数 r 都有一个事件集合 A_r 与其对应，其中 $A_r = \{e : X(e) \leq r\}$ ，则将 X 称作随机变量。

通俗地说，一个随机试验可能结果（基本事件）的全体组成一个基本空间 Ω ，随机变量 X 是定义在基本空间 Ω 上的取值为实数的函数，即基本空间 Ω 中每一个点，也就是每个基本事件都有实轴上的点与之对应。比如：在一次扔硬币事件中，将获得的国徽的次数作为随机变量 X ，则 X 可以取两个值，分别是 0 和 1。

如果随机变量 X 的取值是有限的或是以下可数无穷尽的值：

$$X = \{x_1, x_2, x_3, \dots\}$$

则称 X 为离散随机变量。

如果 X 由全部实数或者由一部分区间组成：

$$X = \{x | a \leq x \leq b\}, -\infty < a < b < \infty$$

则称 X 为连续随机变量，连续随机变量的值是不可数并无穷尽的。

设 P 为概率测度, X 为随机变量, 则函数 $F(x)=P(X\leqslant x)(X\in R)$ 称为 X 的概率分布函数。将随机变量区间的下限定义为 a , 上限定义为 b , 则分布概率 $P(a < X \leqslant b)$ 的定义为:

$$\begin{aligned} p(a < X \leqslant b) &= P(X \leqslant b) - P(X \leqslant a) \\ &= F(b) - F(a) \end{aligned}$$

(2) 离散随机变量分布。离散型随机变量 X 的所有取值与其对应的概率间的关系, 称为离散型随机变量 X 的概率分布, 或称为概率函数。可用列表法表示离散随机变量的分布, 如表 5-1 所示。

表 5-1 离散随机变量分布

随机变量 X 值	x_1	x_2	...	x_n	...
随机变量 X 出现的概率 p	p_1	p_2	...	p_n	...

表 5-1 中第 1 行是随机变量的值, 第 2 行是每个值出现的概率。比如: 用随机变量描述投掷一枚骰子点数的概率分布, 用随机变量 X 表示投掷一枚骰子出现的点数, 概率分布 $p(X=k)=\frac{1}{6}$ ($k=1, 2, \dots, 6$), 可用如表 5-2 所示的列表描述。

表 5-2 骰子点数的概率分布

骰子点数 X	1	2	3	4	5	6
骰子点数 X 出现的概率 p	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

离散随机变量的概率分布具有以下两个性质:

$$p_k \geqslant 0 \quad (k=1, 2, \dots, n)$$

$$\sum_{k=1}^n p_k = 1$$

(3) 0-1 分布。伯努利试验是只有两种可能结果的单次随机试验, 即随机变量 X 只有两个值 0 或 1, 0-1 分布又名伯努利分布或者两点分布, 是一个离散型概率分布。若伯努利试验成功, 则随机变量取值为 1, 否则随机变量取值为 0, 成功概率为 $p(0 \leqslant p \leqslant 1)$, 失败概率为 $q=1-p$ 。0-1 分布的概率质量函数 (概率质量函数的值就是离散随机变量分布的概率) 定义如下:

$$P(X=k)=p^k(1-p)^{1-k}, k=0, 1$$

(4) 二项分布。二项分布即重复 n 次独立的伯努利试验, 在每次试验中只有两种可能的结果, 而且两种结果发生与否互相对立, 并且相互独立, 与其他各次试验结果无关。每次试验的成功概率为 p , 当 $n=1$ 时, 二项分布就是 0-1 分布。

如果随机变量 X 服从参数为 n 和 p 的二项分布, 可记为 $X \sim B(n, p)$, n 次试验正好得到 k 次成功的概率为下面的概率质量函数:

$$P(X=k)=C_n^k p^k(1-p)^{n-k}, k=0, 1, \dots, n$$

其中, $C_n^k = \frac{n!}{k!(n-k)!}$ 。

(5) Poisson 分布。Poisson 分布又名泊松分布, 主要描述单位时间内随机事件发生的次数的概率分布。比如: 某服务设施在一定时间内收到的服务请求的次数、电话交换机接到呼叫的次数、机器出现的故障次数等。泊松分布的概率质量函数为:

$$P(X=k) = e^{-\lambda} \frac{\lambda^k}{k!}, k=0, 1, 2, \dots$$

若 X 服从参数为 λ 的泊松分布, 记为 $X \sim \pi(\lambda)$, 或记为 $X \sim P(\lambda)$ 。

在了解了概率基础知识后, 再来看看连续型随机变量。对于随机变量 X , 若存在非负可积函数 $p(x)$ ($-\infty < x < +\infty$), 使得对任意实数 a, b ($a < b$), 都有

$P(a < X \leq b) = \int_a^b p(x)dx$, 则 X 为连续型随机变量, $P(a < X \leq b)$ 为累积分布函数 (概率密度函数的积分, 可完整描述实随机变量 X 的概率分布), 函数 $p(x)$ 为随机变量的概率密度函数, $p(x)$ 的图像为概率密度曲线, 如图 5-1 所示。

概率密度函数 $p(x)$ 有如下性质:

□ $p(x) \geq 0$

□ $\int_{-\infty}^{+\infty} p(x)dx = 1$

定积分的几何意义为: 对于一个给定的正实值函数 $f(x)$, $f(x)$ 在一个实数区间 $[a, b]$ 上的定积分 $\int_a^b f(x)dx$ 是在 Oxy 坐标平面上, 由曲线 $(x, f(x))$ 、直线 $x=a, x=b$ 以及 x 轴围成的曲边梯形的面积值, 如图 5-2 中的阴影部分。

根据图 5-2 所示的定积分几何意义, $\int_a^b p(x)dx$ 为图 5-2 中的阴影部分, 其中 $p(x)$ 为概率密度函数, 而 $P(a < X \leq b) = \int_a^b p(x)dx$, 因此, 连续随机变量 X 在 a 与 b 之间分布的概率为概率密度函数在区间 $[a, b]$ 上的定积分。

概率质量函数和概率密度函数不同之处在于: 概率质量函数针对离散随机变量定义, 本身代表随机变量的概率; 概率密度函数针对连续随机变量定义, 本身不是概率, 只有对连续随机变量的概率密度函数在某区间内进行积分 (积分后得到累积分布函数) 后才能完成概率的计算。

(6) 连续型均匀分布。连续型随机变量在等长度的每个空间上取值的概率都相同, 则称 X 服从 $[a, b]$ 上的均匀分布, 记作 $X \sim U[a, b]$, 它的概率密度函数为:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{其他} \end{cases}$$

累积分布函数为:

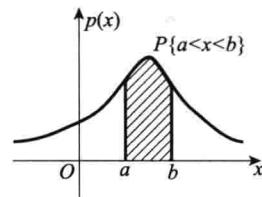


图 5-1 概率密度曲线

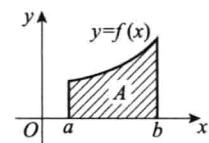


图 5-2 定积分的几何意义

$$F(x) = \begin{cases} 0 & , x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1 & , x \geq b \end{cases}$$

连续型均匀分布函数的期望值和中值等于区间 $[a, b]$ 上的中间点：

$$E[X] = \frac{a+b}{2}$$

方差为：

$$\text{Var}[X] = \frac{(b-a)^2}{12}$$

(7) 指数分布。指数分布可用来表示独立随机事件发生的时间间隔，比如旅客进机场的时间间隔等。它的概率度函数随着取值的变大而指数减小，定义如下：

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0 & , \text{其他} \end{cases}$$

其中， λ 是指每单位时间发生该事件的次数， $\lambda > 0$ 。

累积分布函数定义为：

$$F(x) = \begin{cases} 0 & , x < 0 \\ 1 - e^{-\lambda x}, & x \geq 0 \end{cases}$$

(8) 正态分布。若随机变量 X 服从一个位置参数为 μ 、尺度参数为 σ 的概率分布，记为： $X \sim N(\mu, \sigma^2)$ ，服从正态分布的随机变量的概率规律为：与 μ 越邻近的值的概率越大，离 μ 越远的值的概率越小； σ 越小，分布越集中在 μ 附近， σ 越大，分布越分散。正态分布的概率密度函数曲线呈钟形，又经常被称为钟形曲线，密度函数为：

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad -\infty < x < +\infty$$

累积分布函数为：

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt$$

正态分布密度具有如下性质：

- 曲线关于直线 $x=\mu$ 对称。
- 当 $x=\mu$ 时， $f(x)$ 取得最大值 $\frac{1}{\sqrt{2\pi}\sigma}$ ，
- $f(x)$ 有渐近线 $y=0$ 。
- $f(x)$ 有两个拐点 $(\mu \pm \sigma, \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}})$ 。
- $\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1$ 。

3. 随机变量数字特征相关公式

分布函数完全刻画了随机变量取值的概率规律，因此，在实践中，我们需要知道随机变

量的某些分布指标，比如变量的分布趋势、分布均匀程度等，以便分析随机变量的数字特征。

(1) 数学期望。数学期望反映了随机变量的平均取值。离散性随机变量的数学期望是试验中每次可能结果的概率乘以其结果的总和，离散型随机变量 X 的数学期望为：

$$E[X] = \sum_{k=1}^{+\infty} x_k p_k$$

连续型随机变量 X 的数学期望为：

$$E[X] = \int_{-\infty}^{+\infty} x f(x) dx$$

(2) 方差。方差刻画了随机变量对它的均值的偏离程度，如果 $E[X]$ 是随机变量 X 的期望值（平均数 $\mu=E[X]$ ），则随机变量 X 或者分布 F 的方差为：

$$\text{Var}(X)=E[(X-\mu)^2]$$

(3) 协方差。协方差表示的是两个变量的总体的误差，如果两个变量的变化趋势一致，也就是说，如果其中一个大于自身的期望值，另外一个也大于自身的期望值，那么两个变量之间的协方差就是正值。如果两个变量的变化趋势相反，即其中一个大于自身的期望值，另外一个却小于自身的期望值，那么两个变量之间的协方差就是负值。如果 X 与 Y 是统计独立的，那么二者之间的协方差就是 0。

期望值分别为 $E(X)=\mu$ 与 $E(Y)=\nu$ 的两个实数随机变量 X 与 Y 之间的协方差，定义为：

$$\text{cov}(X, Y)=E((X-\mu)(Y-\nu))$$

其中 E 是期望值，也可表示为：

$$\text{cov}(X, Y)=E((X \cdot Y)-\mu\nu)$$

5.3 回归分析

在进行回归分析时，通常利用数理统计中的回归方法，确定两种或两种以上变量间相互依赖的定量关系。

5.3.1 单变量线性回归

在第 4 章中曾谈到数据的线性回归，可能会出现一种情况，所有的数据点都准确地落在了回归线上。但在现实中，很难有如此精确的模型，比如有两个变量 x 和 y ，其中， $x=[5,7,9,11,16,20]$, $y=[1,2,3,4,7,9]$ ，现在要在 x 与 y 之间建立回归模型，该如何实现？

先用 R 语言构造数据点。

```
> y<-c(5,7,9,11,16,20)
> x<-c(1,2,3,4,7,9)
```

在建立回归线之前，先通过 R 语言的 `plot` 函数绘制这些点的位置，观察分布规律，如图 5-3 所示。

```
> plot(x,y)
```

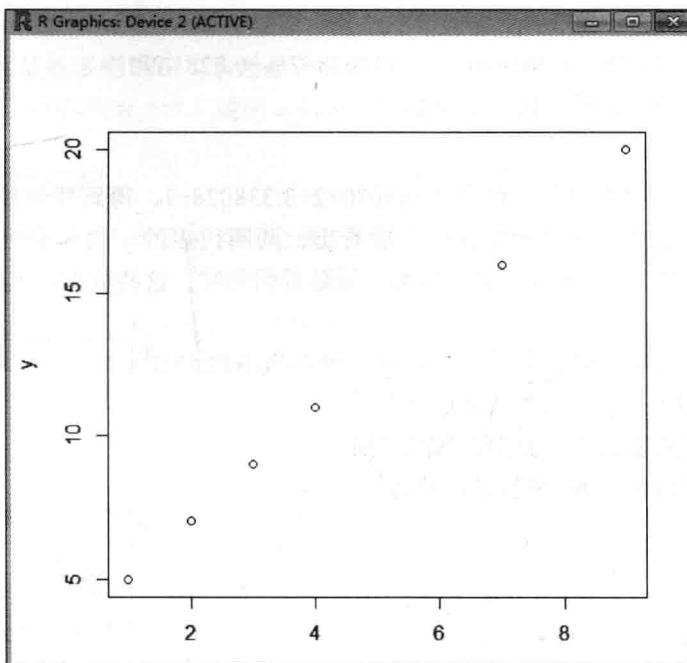


图 5-3 散点图

图 5-3 称为散点图，能清晰地在坐标系中查看数据点位置，横轴为 x ，纵轴为 y 。

从散点图上可以看出 x 和 y 之间存在线性关系，可以表示为 $y = kx + b$ 这种一次函数的模型。不过想要画一条直线完全穿过这些点是不可能的，但能保证这些点到这条直线的距离最小，这个距离就是残差。残差是观测值与预测值之间的差。下面来具体看一下。

首先，通过 `lsfit` 函数计算回归直线方程的斜率和截距以及残差。代码如下：

```
> lsfit(x, y)
$coefficients
Intercept      x
3.338028  1.845070
$residuals
[1] -0.18309859 -0.02816901  0.12676056  0.28169014 -0.25352113  0.05633803
```

上面代码中出现的 `residuals` 表示残差，残差分别反映了这些点与直线的差异，残差越小越好。残差直接反映了数据点到回归直线的距离， -0.18309859 、 -0.02816901 、 0.12676056 、 0.28169014 、 -0.25352113 、 0.05633803 分别是当 x 值为 $[1, 2, 3, 4, 7, 9]$ 时，用 $y = 1.845070x + 3.338028$ 回归方程求得的 y 值与实际 y 值 $[5, 7, 9, 11, 16, 20]$ 的差额。残差 e_i 的计算公式为：

$$e_i = y_i - \hat{y}_i \quad (i=1, 2, \dots, n)$$

其中， y_i 表示实际值， \hat{y}_i 为按回归方程预测的值。

例如，将 $x=2$ 代入该例中的回归方程，即为 $y=1.845070 \times 2 + 3.338028$ ，计算可得到 y 的

预测值，然后得到残差，如下所示：

实际 y 值 - 预测 y 值 $\approx 1.845070 \times 2 + 3.338028 - 7 \approx -0.02816901$

然后，绘制散点图及回归线。命令如下：

```
> abline(lmfit(x, y))
```

刚才将 2 代入回归方程后，计算 $1.845070 \times 2 + 3.338028 - 7$ ，得到残差仅为 -0.02816901 ，在图 5-4 中找到回归线上 $x=2$ 时 y 的值，能看出，圆圈代表的 y 值几乎贴近回归线。再看 $x=1, 3, 4, 7, 9$ 时，回归线对应的 y 值都比较贴近回归线，这些数据点紧靠在回归线周围，显然回归效果不错。

残差不应该有某种趋势，若残差中出现一种明显的趋势，则意味着模型不适合。如图 5-4 所示的残差分布较均匀，没有明显的趋势显示大量数据点偏离了回归线。

事实上，也可以使用 lm 函数进行更详细的回归分析。代码如下：

```
> x<-c(1,2,3,4,7,9)
> y<-c(5,7,9,11,16,20)
> lm(y~x) > xy
> summary(xy)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

1	2	3	4	5	6
-0.18310	-0.02817	0.12676	0.28169	-0.25352	0.05634

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	3.33803	0.16665	20.03	3.67e-05 ***
x	1.84507	0.03227	57.17	5.60e-07 ***

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.222 on 4 degrees of freedom
 Multiple R-squared: 0.9988, Adjusted R-squared: 0.9985
 F-statistic: 3269 on 1 and 4 DF, p-value: 5.604e-07
`> plot(x, y)`
`> abline(lm(y~x))`

在上述代码中，Coefficients 栏的内容如下。

- Estimate: 斜率与截距的估计值。
- Std. Error: 斜率与截距的估计标准差。
- t value: 斜率与截距的假设检验的 t 值。
- Pr(>|t|): 与显著性水平比较，决定是否接受该假设检验。

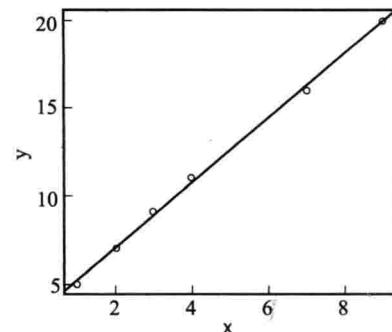


图 5-4 散点图及回归线

在 Coefficients 每行的最后一列有星号，这个星号的含义表示线性关系是否显著：* 的数量是 0~3，* 的数量越多则线性关系越显著。本例中，Coefficients 栏的 Pr(>|t|) 字段（3.67e-05 以及 5.60e-07）后标注有 ***, 说明 x 与 y 之间的线性关系很强。

5.3.2 多元线性回归

多元性回归可建立多个自变量和应变量之间的关系，其回归模型方程一般为：

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_k x_k + e$$

其中， y 为因变量， x_1, x_2, \dots, x_k 为自变量， b_0 为常数项， b_1, b_2, \dots, x_k 为回归系数。

在进行多元线性回归分析时，可使用 lm 函数，在上节例子的基础上增加一个自变量 $x2=[3,4,5,6,8,10]$ ，来看看会有什么效果。代码如下：

```
> y<-c(5,7,9,11,16,20)
> x<-c(1,2,3,4,7,9)
> x2<-c(6,8,10,12,16,20)
> lm(y~x+x2)->xy2
> summary(xy2)
Call:
lm(formula = y ~ x + x2)

Residuals:
    1         2         3         4         5         6 
-7.495e-16  9.195e-16  4.172e-17 -2.117e-16  1.839e-16 -1.839e-16 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.000e+00 3.787e-15 2.640e+14 <2e-16 ***
x           1.000e+00 1.359e-15 7.357e+14 <2e-16 ***
x2          5.000e-01 8.019e-16 6.236e+14 <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.121e-16 on 3 degrees of freedom
Multiple R-squared:      1,   Adjusted R-squared:      1 
F-statistic: 1.591e+32 on 2 and 3 DF,  p-value: < 2.2e-16
```

通过上述回归分析，得出该回归方程为 $y=x\times 0.5\times x2+1$ ，Std. Error、t value、Pr(>|t|) 以及线性相关程度等指标的含义同单变量线性回归类似。

5.3.3 非线性回归

非线性回归模型较多，其中应用得较多的有以下模型：

(1) 多项式模型：

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_k x^k + \varepsilon$$

(2) 指数模型：

$$y = a e^{bx} \varepsilon$$

(3) 幂函数模型:

$$y=ax_1^{b1}x_2^{b2}\varepsilon$$

(4) 成长曲线模:

$$y=1/(\beta_0+\beta_1 e^{-x}+\varepsilon)$$

实际应用中,除上述模型外,还有很多非线性回归模型,但无论是哪种非线性回归模型,最后都可以通过变量变换转化为线性模型,从而用最小二乘法进行回归分析。下面以 $y=\beta_0+\beta_1 e^{bx}+\varepsilon$ 模型为例讲解非线性回归的方法。

(1) 准备回归分析用数据(假设已预先知道回归方程,通过回归方程精确计算 y 值),

```
>x<-c(1,2,3,4,7,8,9)
>y <- 100 + 10 * exp(x / 2) + rnorm(x)
```

(2) 使用 R 语言的 nls 函数,应用最小二乘法原理,实现非线性回归分析。

```
>nlmod <- nls(y ~ Const + A * exp(B * x))
```

(3) 使用 summary 函数分析拟合结果。

```
> summary(nlmod)

Formula: y ~ Const + A * exp(B * x)

Parameters:
            Estimate Std. Error t value Pr(>|t|)
Const 1.0001e+02 8.377e-01 119.4 2.95e-08 ***
A      1.006e+01 1.759e-01   57.2 5.59e-07 ***
B      4.995e-01 1.925e-03   259.5 1.32e-09 ***

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.1 on 4 degrees of freedom
```

Number of iterations to convergence: 8
Achieved convergence tolerance: 4.887e-07

>

其中,在 Parameters 栏,对方程涉及的以下 3 个参数进行了预测:

$$\beta_0=\text{Const}=1.001\text{e+02}$$

$$\beta_1=A=1.006\text{e+01}$$

$$b=B=4.995\text{e-01}$$

将以上参数与程序中 y 值的生成规则进行对比,可以看见,拟合效果还是不错的。

前面为解释非线性回归过程,将 x 代入参数确定的非线性回归方程计算 y 值,然后,依据 x 和 y 推出非线性回归方程的参数。但实践应用中,往往需要依据一组 x 值和 y 值,推导非线性方程的参数,因此,尝试通过 rnorm 函数产生较小的随机数,加在精确计算的 y 值上,这样计算后形成的非线性回归模型拥有一定的残差,较接近真实环境。

```
>y <- 100 + 10 * exp(x / 2) + rnorm(x)
```

(4) 绘制拟合效果图。代码如下：

```
>plot(x,y, main = "nls(o)")
>curve(100 + 10 * exp(x / 2), col = 4, add = TRUE)
>lines(x, predict(nlmod), col = 2, type='b')
```

如图 5-5 所示为拟合效果图，显示出拟合效果不错。其中，偏上的线为预测的回归线，偏下的线为实际方程。回归在 [4,7] 的区间内拟合效果较差，这是样本数据太少的原因，因为样本数据仅有 9 个。

参与拟合的样本数据量决定了拟合效果的好坏。可以加大自变量的数量，重新应用 nls 函数进行拟合，来看看效果如何。代码如下：

```
>x<-seq(1,10,0.1)

>y <- 100 + 10 * exp(x / 2) + rnorm(x)

>nlmod <- nls(y ~ Const + A * exp(B * x))

>plot(x,y, main = "nls(o)")

>curve(100 + 10 * exp(x / 2), col = 4, add = TRUE)

>lines(x, predict(nlmod), col = 2)
```

如图 5-6 所示是相应的效果图，从中可以看出，回归线与实际方程线很接近。

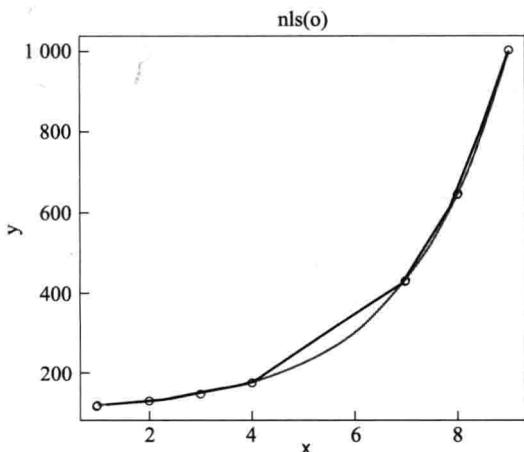


图 5-5 拟合效果图

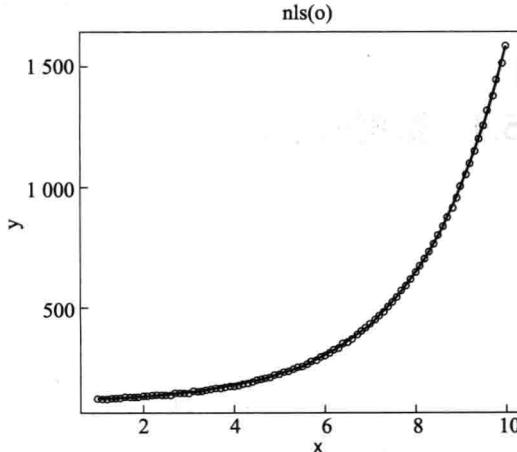


图 5-6 非线性回归效果图

不过，图 5-6 所示的回归效果仍存在一个问题，就是样本过于集中在回归线上了，为使回归分析更接近真实应用环境，需要继续加大随机数的范围，增加非线性回归的残差，使样本点散布在回归线周围。

```
>x<-seq(1,10,0.1)
>y <- 100 + 10 * exp(x / 2) + rnorm(x)*100
>nlmod <- nls(y ~ Const + A * exp(B * x))
```

```
>plot(x,y, main = "nls(o)")
>curve(100 + 10 * exp(x / 2), col = 4, add = TRUE)
>lines(x, predict(nlmod), col = 2)
```

从图 5-7 可以看出，样本点虽然没有集中在回归线上，而是散落在回归线周围的区域，产生的残差较大，但样本点的整体走向与回归线一致，此外，回归线与实际方程这两条线几乎重叠，说明回归分析较准确地预测出回归方程的各个参数。因此，从整体上观察，拟合效果不错，回归模型适当。

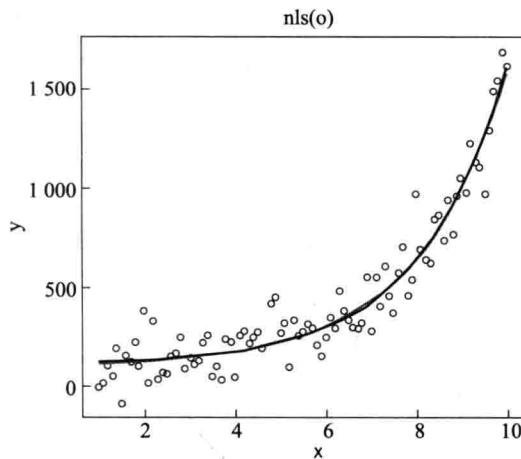


图 5-7 接近真实环境的非线性回归

5.4 数据分析基础

5.4.1 区间频率分布

下面是美国地震台网公布的全球 2013 年 5 月 20 日 22 点到 24 点发生的所有地震的震级。

时间	震级
2013-05-20T23:57:12.000+00:00	1.6
2013-05-20T23:57:12.000+00:00	0.9
2013-05-20T23:52:59.000+00:00	2.1
2013-05-20T23:49:15.100+00:00	2.2
2013-05-20T23:46:36.000+00:00	2.3
2013-05-20T23:44:07.000+00:00	1.7
2013-05-20T23:38:17.000+00:00	1.3
2013-05-20T23:34:12.400+00:00	1.6
2013-05-20T23:33:43.440+00:00	4.7
2013-05-20T23:25:20.500+00:00	1.2
2013-05-20T23:23:35.100+00:00	0.9

2013-05-20T23:07:34.960+00:00	4.7
2013-05-20T23:06:42.800+00:00	0.6
2013-05-20T23:01:25.480+00:00	5.3
2013-05-20T22:59:58.000+00:00	1.1
2013-05-20T22:51:47.120+00:00	4.8
2013-05-20T22:48:40.570+00:00	4
2013-05-20T22:48:18.350+00:00	4.2
2013-05-20T22:36:27.310+00:00	4.6
2013-05-20T22:13:36.000+00:00	1.3
2013-05-20T22:13:09.000+00:00	2.1
2013-05-20T22:10:47.000+00:00	1.5
2013-05-20T22:09:33.600+00:00	3

下面就以上面的数据为例，来讲述区间频率分布。现在的任务是完成地震震级分析。

(1) 将地震震级数据放入一个向量中。代码如下：

```
>mag<-c(1.6,0.9,2.1,2.2,2.3,1.7,1.3,1.6,4.7,1.2,0.9,4.7,0.6,5.3,1.1,4.8,4,4.2,4,
6,1.3,2.1,1.5,3)
> mag
[1] 1.6 0.9 2.1 2.2 2.3 1.7 1.3 1.6 4.7 1.2 0.9 4.7 0.6 5.3 1.1 4.8 4.0 4.2
[19] 4.6 1.3 2.1 1.5 3.0
```

(2) 使用 cut 函数将震级分成 5 个区间，并建立因子。代码如下：

```
>factor(cut(mag,5))
[1] (1.54,2.48]  (0.595,1.54]  (1.54,2.48]  (1.54,2.48]  (1.54,2.48]
[6] (1.54,2.48]  (0.595,1.54]  (1.54,2.48]  (4.36,5.3]   (0.595,1.54]
[11] (0.595,1.54] (4.36,5.3]   (0.595,1.54] (4.36,5.3]   (0.595,1.54]
[16] (4.36,5.3]   (3.42,4.36]  (3.42,4.36]  (4.36,5.3]   (0.595,1.54]
[21] (1.54,2.48]  (0.595,1.54] (2.48,3.42]  (3.42,4.36] (4.36,5.3]
Levels: (0.595,1.54] (1.54,2.48] (2.48,3.42] (3.42,4.36] (4.36,5.3]
```

(3) 统计因子频率。代码如下：

```
>factor(cut(mag,5))->magfactor
> table(magfactor)
magfactor
(0.595,1.54]  (1.54,2.48]  (2.48,3.42]  (3.42,4.36]  (4.36,5.3]
8             7             1             2             5
```

可以看出 2013 年 5 月 20 日 22 点到 24 点期间，全球发生的地震在 (0.595,1.54] 内有 8 起，在 (1.54,2.48] 内有 7 起等。

(4) 绘制直方图。在 R 语言中，可使用 hist 函数绘制直方图。

```
> hist(mag, breaks=5)
```

绘制结果如图 5-8 所示。

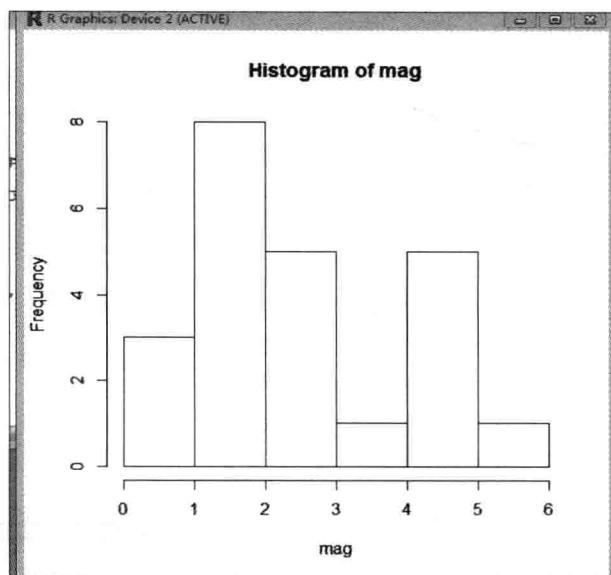


图 5-8 直方图

5.4.2 数据直方图

直方图又称柱状图、质量分布图，是一种对数据分布情况的图形表示，由一系列高度不等的纵向条纹或线段表示数据分布的情况，它根据从生产过程中收集来的质量数据分布情况，组成以组距为底边、以频数为高度的一系列连接起来的直方型矩形图。

地震数据是一种次数直方图，是由若干宽度相等、高度不一的直方条紧密排列在同一基线上构成的图形，其中基线上每个区间代表了一段震级，高度代表了这段震级发生地震的次数（频率）。

(1) 用 R 语言的 `read.table` 函数读取地震震级的文件 (`read.table` 方法可读取文件，生成 `list` 组件类型的数据集，并且，通过指定 `header=TRUE` 参数，可将文件头作为字段变量名)。

```
> read.table("eqweek.csv", header=TRUE, sep=",")->earthquake
```

(2) 显示读取的地震数据 (2013.5.14~2013.5.20)，验证读取内容是否完整。

```
> earthquake      DateTime.Latitude.Longitude.Depth.Magnitude.MagType.NbStations.
Gap.Distance.RMS.Source.EventID.Version
 1          2013-05-20T23:57:12.000+00:00,63.45,-148.291,5.5,1.6,M1,,,,
.8,ak,ak10720946,1.3691E+12
 2          2013-05-20T23:52:59.000+00:00,61.337,-152.069,81.4,2.1,M1,,,
5,ak,ak10720941,1.36909E+12
 3          2013-05-20T23:49:15.100+00:00,19.99,-155.426,38.2,2.2,Md,,133,0.1,0.
11,hv,hv60501711,1.3691E+12
 4          2013-05-20T23:46:36.000+00:00,60.498,-142.974,4.2,2.3,M1,,,
3,ak,ak10720934,1.36909E+12
.....
```

(3) 绘制数据直方图, 观察从 2013 年 5 月 14 日至 2013 年 5 月 20 日这周内全球地震震级的分布情况。代码如下:

```
> hist(earthquake$Magnitude, 5)
```

生成的直方图如图 5-9 所示。其中横轴是震级, 纵轴是频率分析。通过观察可得出结论: 震级在 1~2 级的地震发生频率最高, 因为从 x 为 1~2 时, 长方形柱体最高; 2~3 级的柱体高度仅次于 1~2 级, 发生频率排名第二; 发生频率最小的是 6~7 级的地震, 柱体高度很小。总体来说, 1~3 级和 4~5 级的地震发生较频繁。

数据直方图能直观地说明数据在每个区间的分布频率, 但如果需要精确的分布频率, 则需要使用 R 语言的因子对象。先通过 cut 函数将数据分组, 然后通过 factor() 函数生成因子对象, 最后使用 table 函数分析频率。代码如下:

```
> table(factor(cut(earthquake$Magnitude, 5)))
```

(0.995,2.1]	(2.1,3.2]	(3.2,4.3]	(4.3,5.4]	(5.4,6.51]
693	200	46	126	10

观察上述结果的最后两行, 每个区间的震级频率一目了然, $(0.995,2.1]$ 区间的震级频率是 693 次, $(2.1,3.2]$ 区间的震级频率是 200 次等。

5.4.3 数据散点图

数据散点图是指数据点在直角坐标系平面上的分布图, 通过直接观察图形辨认某现象的测量值与可能原因因素之间的关系, 具有快捷、易于交流和易于理解的特点。

数据散点图表示因变量随自变量而变化的大致趋势, 将序列显示为一组点, 值由点在图表中的位置表示, 类别由图表中的不同标记表示。通常用垂直轴表示现象值 Y , 用水平轴表示可能有关系的原因因素 X , 通过对观察分析, 来判断两个变数之间的相关关系。此外, 依据散点图可选择函数对数据点进行拟合, 建立回归模型。

下面继续以全球一周地震数据为例, 来讲解数据散点图。

(1) 将变量放到搜索路径上。代码如下:

```
> attach(earthquake)
```

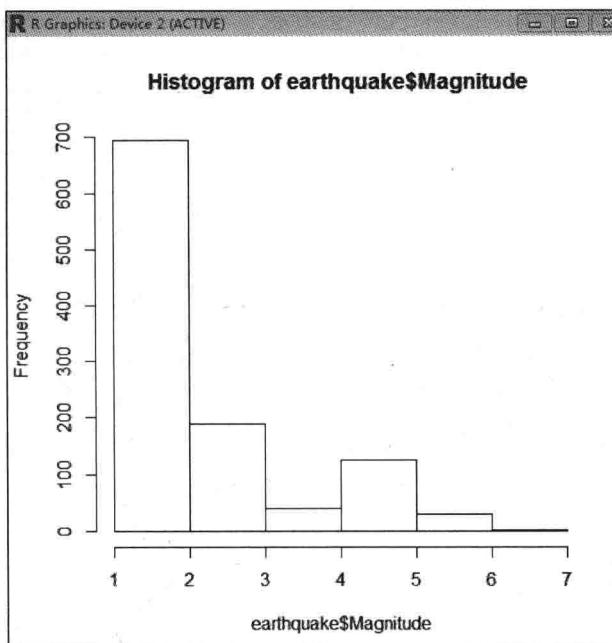


图 5-9 震级分布直方图

(2) 分析地震震深。代码如下：

```
> summary(Depth)
Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
0.10 5.80 12.15 30.82 38.00 630.70 39
```

在上述结果中，Min 表示地震震深的最小值，Max 表示最大值，Median 为中位数，Mean 为平均值。我们试着从下面的散点图中分析一下地震震深与震级的关系，如图 5-10 所示。

在图 5-10 中，Depth 是震深，Magnitude 是震级，从表面上看一周中 Depth 和 Magnitude 之间没有关系，但仔细观察这个图，可发现一个有趣的结果：在这一周里当震深超过 300 后，震级都接近 5 或在 5 以上，而在 300 以内时，震级并不确定。

(3) 对震深的直方图进行分析。先来绘制相关直方图，代码如下：

```
> hist(Depth)
```

绘制的图形如图 5-11 所示。

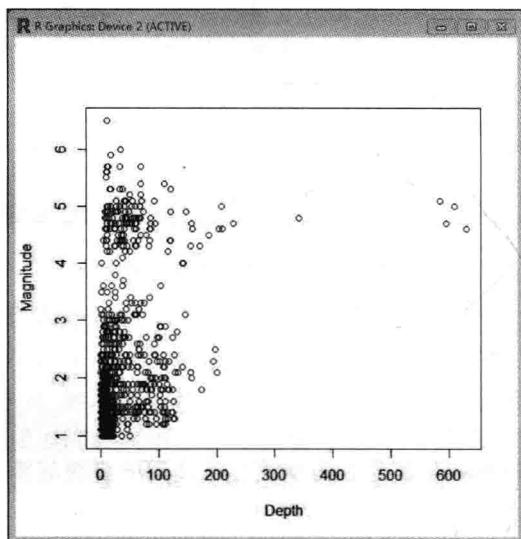


图 5-10 震深与震级关系的散点图

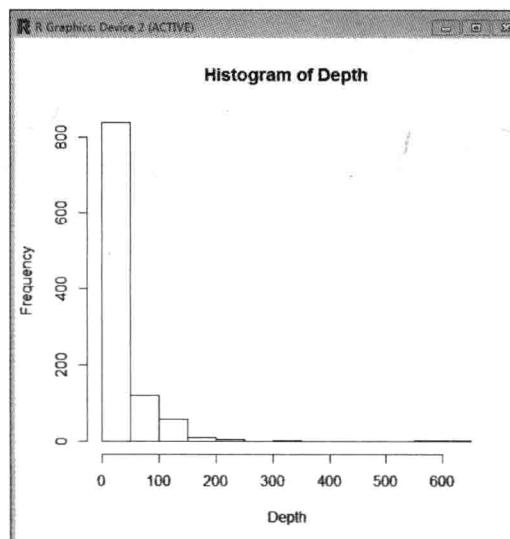


图 5-11 震深的直方图

观察图 5-11 可得出结论：这个星期内发生的绝大部分地震的震深在 100 以内，这个区间的代表频率的柱体高度最高，而发生最少的 350~550 段和 250~300 段，这两个区间的柱体几乎贴近 X 轴，高度很小。

(4) 分析带数据点的震级直方图。为提高直方图的表示能力，有时需要在直方图中显示实际的数据点，通过观察这些数据点的数量，可分析数据点在每个区间的分布密集程度，可通过 R 语言的 rug 函数绘制数据点。代码如下：

```
> hist(Magnitude)
> rug(Magnitude)
```

绘制出的图形如图 5-12 所示。

仔细观察图 5-12 能看出：在 3~4 震级区域内，数据点分布并不均匀，在靠近 3 的区域内，数据更为密集，数据点集中分布在偏向于 3 的区域；此外，在 5~6 级震级区域也出现类似现象，数据点偏向于 5，这说明 3~4 级或 5~6 级震级范围内，大部分地震的震级不是非常大，偏向于 3 或 5。

这些只是根据一个星期的数据进行分析得到的结果，不一定就代表真正的答案。答案要通过分析大量数据以及数据各个指标关系，同时结合地球物理知识的研究才能得出。

由于图形直观性很强，浅显且易于理解，因此在数据分析中，经常需要绘制图表和直线。前面已经讲解过绘图的方法，在此补充一下画线方法，在 R 语言中使用 `lines` 函数完成绘制直线。比如要绘制一个 (10,40)、(20,50)、(30,60) 的散点图，并将点连成线，可如下编写代码：

```
> plot(c(10,20,30),c(40,50,60))
> lines(c(10,20,30),c(40,50,60))
```

绘制出的散点及直线如图 5-13 所示。

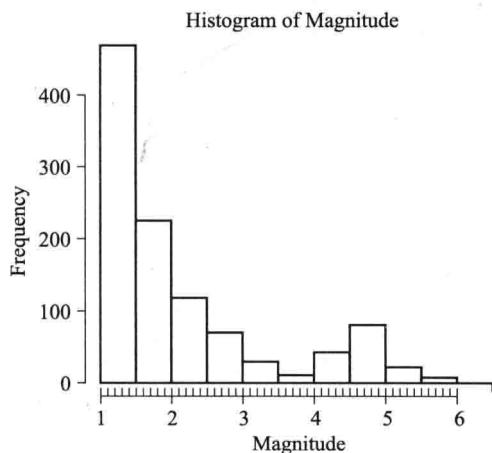


图 5-12 带数据点的震级直方图

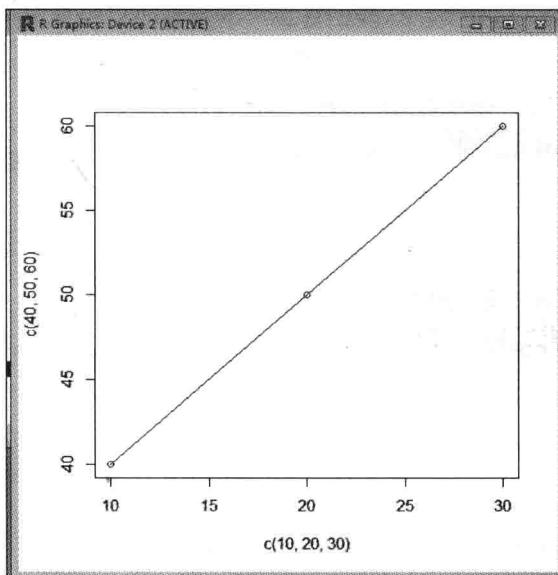


图 5-13 散点及直线图

5.4.4 五分位数

分位数是描述数据位置的一种方法，它将一个随机变量的分布范围分为几个等份的数值点。分位数法被用来识别某临界值，一般情况下可使用分位数（或分位点）描述小于等于这个临界值的观测值数量占整个数据中的某个具体比率（小于等于该值的数据为一给定的比率）。

常用的分位数有中位数、四分位数、五分位数、十分位数、百分位数等，其中，中位数将数据分布范围分成了相等的两个部分。此外，我们还能将数据分布范围分成更小尺寸的分

割线，四分位数将一个分布分成 4 等份，而五分位数将其分成 5 等份，十分位数将其分成 10 等份，百分位数将其分成 100 等份等。

五分位数法是数据分析的常用方法，在 R 语言中，可使用 `fivenum` 函数计算五分位数（`fivenum` 函数会返回以下数据：minimum、lower-hinge、median、upper-hinge、maximum）。下面用 `fivenum` 函数对地震数据进行分析。代码如下：

```
> fivenum(Magnitude)
[1] 1.0 1.3 1.7 2.5 6.5
```

分析结果表明，震级最小为 1.0，最大为 6.5，中位数为 1.7，通过 1.7 将一组数据分为上下两组，然后再计算上下两组的中位数 1.3 与 2.5。

5.4.5 累积分布函数

累积分布函数完整地描述出了一个实数随机变量 X 的概率分布，是概率密度函数的积分，与概率密度函数相对。它被定义为随机变量小于或者等于某个数值的概率 $P(X \leq x)$ ，即 $F(x) = P(X \leq x)$ 。我们计算一下震级的累积分布。

(1) 通过 R 语言的 `ecdf` 函数计算累积分布。

```
> ecdf(Magnitude) -> mag_ecdf
> ecdf
Empirical CDF
Call: ecdf(Magnitude)
x[1:50] = 1, 1.1, 1.2,
..., 6, 6.5
```

(2) 通过 `plot` 函数绘制累积概率图，直观展示震级累积分布。

```
> plot(mag_ecdf, do.points=FALSE,
verticals=TRUE)
```

绘制出的图形如图 5-14 所示。

图 5-14 的 x 轴是震级， y 轴是累积分布概率，当震级升高时，累积分布概率也随之提高，这个趋势很正常，因为累积分布概率是指小于或等于某个震级的概率。比如：从图 5-14 观察，发生地震的震级在 3 以内的概率接近 80%，震级在 2 以内的概率接近 60%。

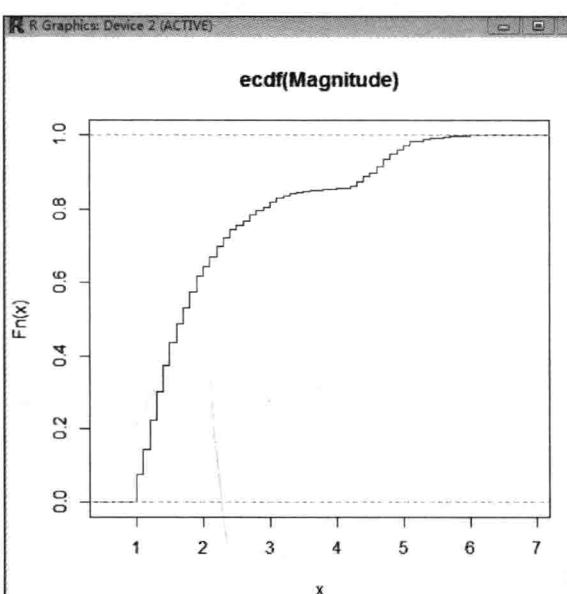


图 5-14 震级累积分布概率

5.4.6 核密度估计

1. 核密度原理

利用前面所说的直方图估计数据分布密度是有局限性的，因为数据分布的密度函数是不

平滑的，它受子区间宽度影响较大，当数据维数超过二维时就有局限。

什么是密度函数呢？连续型随机变量的概率密度函数（不至于混淆时可简称为密度函数）是一个描述这个随机变量的输出值，在某个确定的取值点附近的可能性的函数，而随机变量落在某个区域之内的概率为概率密度函数在这个区域上的积分，当概率密度函数存在的时候，累积分布函数是概率密度函数的积分。

如何由给定样本点集合求解随机变量的分布密度函数？解决这一问题的方法包括参数估计和非参数估计。参数估计中常用的是参数回归分析，它假定数据分布符合某种特定的性态，如线性、可化线性或指数性态等，然后确定回归模型的未知参数。但参数模型的这种基本假定与实际的物理模型之间常常存在较大的差距，于是 Rosenblatt 和 Parzen 提出了核密度估计方法，它可估计未知的密度函数，是非参数估计方法之一。

假设样本数据值在 D 维空间服从一个未知的概率密度函数，那么在区域 R 内的概率为：

$$P = \int_R p(x) dx$$

上式中，P 是每个样本数据点落入区域 R 的概率，假设在 N 个样本数据点中，有 K 个落入了区域 R，那么就应该服从二项分布。公式如下：

$$\text{Bin}(K|N,P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{1-K}$$

在 N 的样本数据很大时，K 约等于 N×P。而另一方面，假设区域 R 足够小，那么 P 约等于 $p(x) \times V$ (V 为区域 R 的空间)。

于是，结合两个不等式子可得：

$$p(x) = \frac{K}{NV} \quad (5-1)$$

根据上式，来估算 $p(x)$ 就有两种方式：第一，K 不变，通过决定区域 V 的大小来估算密度函数，采用 K-nearer-neighbour 方法；第二，V 不变，通过决定 K 的大小来估算密度函数，采用 kernel 方法。这里选择第二种方式，假设区域 R 是一个以 x 为中心、边长为 h 的极小立方体（也就是 V 不变），定义 kernel 函数（数据维数为 D 维，当样本数据点落入小立方体时，函数值为 1，其他情况下为 0）的公式如下：

$$k(u) = \begin{cases} 1, & |u_i| \leq 1/2, \quad i=1, \dots, D \\ 0, & \text{其他} \end{cases}$$

落入立方体数据点的总个数 K 就可以表示为：

$$K = \sum_{n=1}^N k\left(\frac{x-x_n}{h}\right) \quad (5-2)$$

根据式 (5-1)，把式 (5-2) 代入式 (5-1) 中，可得：

$$p(x) = \frac{1}{N} \sum_{n=1}^N \frac{1}{h^D} k\left(\frac{x-x_n}{h}\right)$$

上式中， $V=h^D$ 。

2. 核密度计算

R 语言可使用 density 函数进行核密度估计，density 函数默认情况下在 512 个点上估计密度值。下面就用它来计算一下地震数据中的震级核密度。首先，指定 hist 函数的 prob 参数为 TRUE，绘制核密度直方图；然后通过 lines 函数，绘制核密度曲线。代码如下：

```
> hist(Magnitude, prob=TRUE)
> lines(density(Magnitude))
```

绘制出的核密度如图 5-15 所示，图中曲线就是核密度曲线。

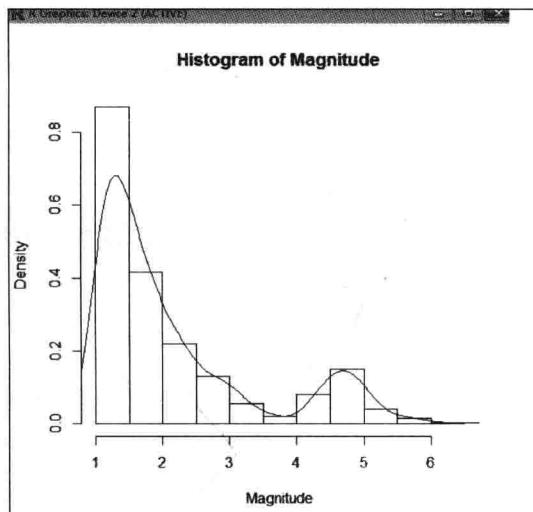


图 5-15 震级核密度

5.5 数据分布分析

这里以老年常见病数据为例，来讲解数据分布的分析。如表 5-3 所示是一些老年常见病数据。

表 5-3 老年常见病数据

年龄	疾病名称	急腹症	肿瘤
71	肺恶性肿瘤	0	1
75	直肠恶性肿瘤	0	1
75	肠梗阻	1	0
71	急性胆管炎	1	0
77	胆管恶性肿瘤	0	1
...

(1) 在 R 中加载数据，然后查看老年病的老人年龄分布情况及概率密度分布，绘制年龄直方图。代码如下：

```
> read.table("aged_patients.csv", header=TRUE, sep=",")->agedpatients
> hist(agedpatients$年龄)
```

绘制出的年龄分布直方图如图 5-16 所示。

(2) 对年龄进行核密度估计，并绘制核密度曲线。

```
> hist(agedpatients$年龄, prob=TRUE)
> lines(density(agedpatients$年龄))
```

绘制出的核密度曲线，如图 5-17 所示。

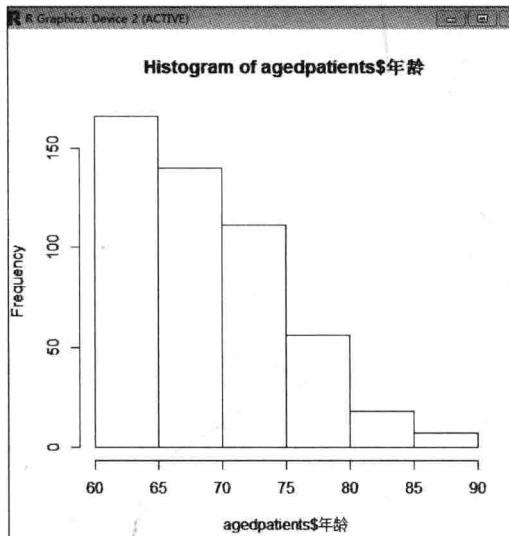


图 5-16 年龄分布直方图

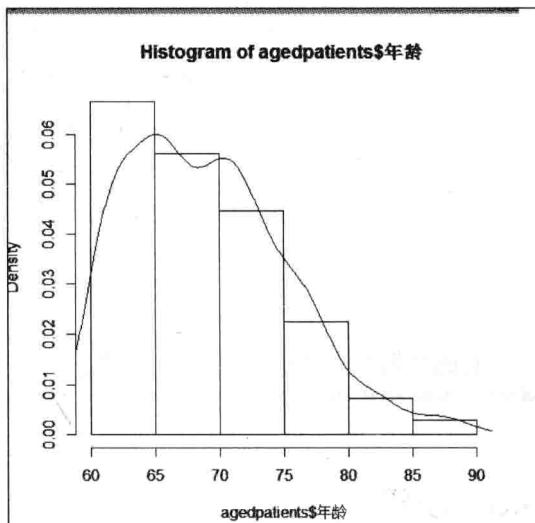


图 5-17 年龄的核密度曲线

(3) 通过 min 和 max 两个函数分析患者的最小年龄和最大年龄，通过 mean 函数分析年龄的平均值，通过 var 函数计算年龄的方差。

```
> min(agedpatients$年龄)
[1] 60
> max(agedpatients$年龄)
[1] 90
> mean(agedpatients$年龄)
[1] 69.09839
> var(agedpatients$年龄)
[1] 38.60801
```

由上述分析可得出一个结论：63~72 岁这个阶段的老人必须要注意身体，坚持运动，保持健康，这个年龄段是急腹症和肿瘤两种老年病的高发期，发生概率较大。从 60 到 90 岁的老人都有可能患上这两种老年病，患者的平均年龄是 69 岁。标准差比较大，看来这两种老年病在老人的各个年龄段中发生得比较普遍。

(4) 按年龄分类汇总肿瘤和急腹症的数量。代码如下：

```
> attach(agedpatients)
```

```

> tapply(肿瘤, 年龄, sum)
   60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 87
88 90
   19 26 26 28 21 35 37 22 21 23 29 26 31 16 19 12 13 19   8   5   4   3   7   2   1   1   0
3  1
> tapply(急腹症, 年龄, sum)
   60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 87
88 90
   2   0   0   2   0   0   1   0   1   2   4   1   1   0   3   1   2   2   0   2   0   1   0   1   0   1   2
0  0

```

(5) 分年龄统计肿瘤患者的数据。代码如下：

```

> table(factor(cut(agedpatients$年龄[agedpatients$肿瘤==1], 5)))
(60,66] (66,72] (72,78] (78,84] (84,90]
155      158      118      22       5

```

(6) 分年龄统计急腹症患者的数据。代码如下：

```

> table(factor(cut(agedpatients$年龄[agedpatients$急腹症==1], 5)))
(60,65.4] (65.4,70.8] (70.8,76.2] (76.2,81.6] (81.6,87]
4           8           8           5           4

```

上述分析结果表明：急腹症患者在 65 岁到 77 岁之间发病率较高，其中 70 岁时发病率最高；而肿瘤患者在 60 岁到 72 岁之间发病率较高，其中 69 岁时发病率最高。

5.6 小结

本章对统计学基础进行阐述，介绍了数据分析的概念、发展以及需要的基础知识，同时从线性回归和非线性回归两个方面讲述了数据分析基本方法——回归分析，并以实例说明了用 R 语言进行数据分布情况分析的方法。

统计分析在机器学习和数据分析中有着举足轻重的地位。李开复在攻读博士期间主攻语音识别。他的导师坚持的方向是发展和完善专家系统。而他最终发现，专家系统是有严重局限性的，无法延伸到做不特定语者的语音识别。他认为有数据支持的统计模式是唯一的希望，于是改用统计的方法来进行语音识别。3 年中，他用统计的方法把语音识别的准确率从 40% 逐步提高到 80%、90%，最后达到了 96%，《商业周刊》把他的发明选为 1988 年最重要的科学发明。他用统计学方法做出的语音识别博士论文至今还被用作语音识别产品的理论基础。

Google 在 2006 年面对广大用户推出了关键词统计分析系统：Google Trends，链接为：<http://www.google.com/trends>。这是一个非常有意思和价值的产品，有兴趣的读者不妨一试。下面是利用这个系统对“machine learning”这个搜索关键词进行的统计分析，从图 5-18 和图 5-19 中可以明显看出，机器学习越来越受人关注，尤其是从 2011 年年底开始，人们对机器学习的热衷度在持续上升，因为这段时期的曲线呈稳步上扬趋势。

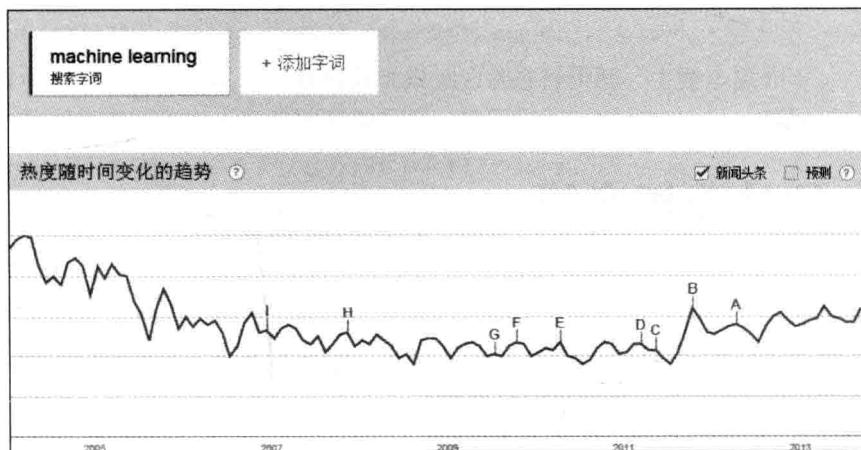


图 5-18 “机器学习”在 Google Trends 中的搜索结果

热门		上升
the learning machine	100	
pattern recognition	50	
machine learning pdf	50	
machine learning algorithms	50	
data mining	45	
machine learning stanford	45	
machine learning classification	40	
		上升
		飙升
		飙升
		+250%
		+250%
		+180%
		+90%

图 5-19 “机器学习”在 Google Trends 中的搜索结果

思考题

1. 本章中介绍了用 R 语言进行回归分析的方法。请对下列几组数据进行回归分析。

- (1) $x=[2,5,8,9,12,15], y=[18,52,78,101,125,148]$
- (2) $x=[2,5,8,9,12,15], y=[5,120,502,739,1708,3415]$



提示

第一组数据适用线性回归，第二组数据适用非线性回归，回归方程为 $y=x\times 20+\exp(x/5)$ 。

2. 在 Google 关键词统计分析中搜索“机器学习”的中文关键词，描述人们对该关键词兴趣的发展趋势，目前机器学习的哪些分支领域正在成为人们关注的热点。

3. 下载本书例子中的美国地震台数据 earthquakes.csv，分析 2013.3~2013.4 期间的数据，分震级统计这段时期全球发生的地震，同时做出这段时间的震深与震级散点图，计算震级的累积分布，并显示累积分布图。

第6章

统计分析案例

本章将以 R 语言为分析工具来剖析统计分析案例，对于其中涉及的统计分析知识也会做简单介绍，请各位读者按准备篇的指导将 R 语言计算平台搭建好。

6.1 数据图形化案例解析

数据是事实，也称观测值，是实验、测量、观察、调查等活动的结果，常以数量的形式给出。数据分析的目的是把隐没在一大批看似杂乱无章的数据中的有用信息集中、萃取和提炼出来，以找出所研究对象的内在规律。

6.1.1 点图

下面以 2010 年全国各行业就业调查数据（部分数据如表 6-1 所示，完整数据在本书下载包中）为依据，以点图分析为手段，剖析电子行业劳动报酬水平。

表 6-1 全国各行业就业调查部分数据

行业代码	行业名称	平均劳动报酬	平均教育经费
10000	农林牧渔业	22475	143
10100	农业	17542	230
10300	畜牧业	27958	308
10310	牲畜的饲养	25636	196
20000	采矿业	43713	44
20800	黑色金属矿采选业	43895	42
...

读入如表 6-1 所示的数据文件。代码如下：

```
> read.table("youxiangz.csv",,header=TRUE,sep=",")->jiuye
```

分析行业报酬水平，以电子行业的劳动报酬为例进行讲解，主要步骤如下。

(1) 从数据集中筛选电子行业的劳动报酬。代码如下：

```
> jiuye$行业名称[grep1("电子",jiuye$行业名称)]->jyhy
```

```
> jiuye$平均劳动报酬[grep1("电子",jiuye$行业名称)]->jygz
```

```
> names(jygz)<-jyhy
```

(2) 绘制电子行业薪水点图，如图 6-1 所示。从图 6-1 中可清楚地看到 7 个电子行业的薪水分布情况。可将图 6-1 分为若干行，每一行代表一个行业，点在每行的不同位置代表不同的报酬，所有行的数值遵循同一刻度（刻度尺在点图的最下方，从 40000 到 120000 分成 4 个区域）。

(3) 找到薪水最高、最低的行业。首先找到图 6-1 中相应行业代表的行，然后在该行找到由点代表的刻度值（点在某行的位置），最后在刻度尺中找到相应数值，读取数值。很明显，代表“电子计算机制造”行业报酬的点在所有行中最贴近右端，属于薪水最高的行业，而“家用电器及电子产品专门零售”行业的数值在最左端，属于电子行业中薪水最低的行业。

```
> dotchart(jygz)
```

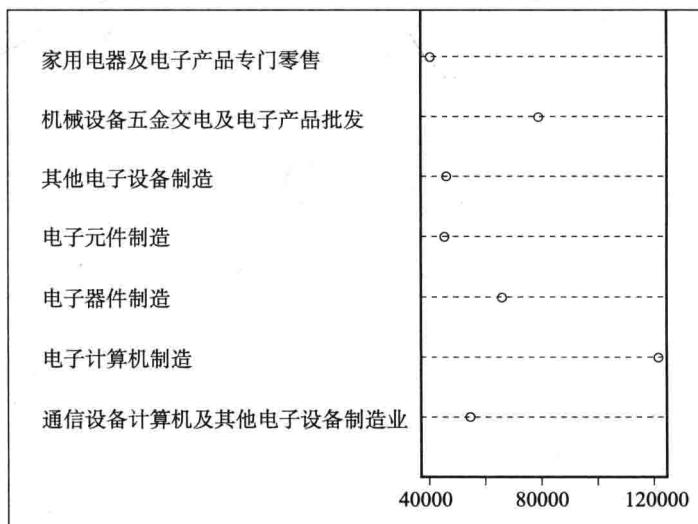


图 6-1 电子行业薪水点图

6.1.2 饼图和条形图

下面以饼图和条形图为分析手段，对中介行业的平均劳动报酬进行剖析。

(1) 以条形图来表示平均劳动报酬。代码如下：

```
> jiuye$平均劳动报酬[grep1("中介",jiuye$行业名称)]->jygz
> jiuye$行业名称[grep1("中介",jiuye$行业名称)]->jyhy
```

```
> names(jygz)<-jyhy
> barplot(jygz,horiz = TRUE)
```

绘制结果如图 6-2 所示。

从图 6-2 可明显观察到，“科技中介服务”行业的平均劳动报酬位居第一，“职业中介服务”位居最后。

(2) 除了条形图，还可以用饼图分析平均劳动报酬。代码如下：

```
> pie(jygz)
```

绘制结果如图 6-3 所示。

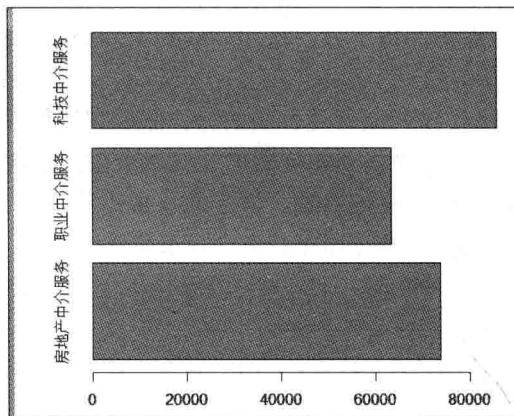


图 6-2 中介行业的平均劳动报酬条形图

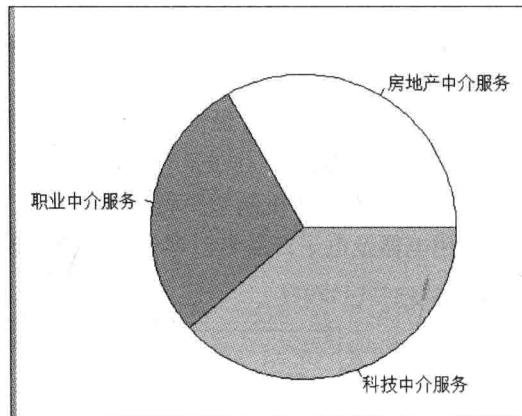


图 6-3 中介行业的平均劳动报酬饼图

观察图 6-3，代表“科技中介服务”的面积所占比重最大，因此属于平均劳动报酬最高的中介服务行业。

6.1.3 茎叶图和箱线图

本节将要讲解的茎叶图和箱线图，可能大家平时接触得比较少，但在数据分析中经常用到。茎叶图和箱线图不同于前面的图表，表面看上去比较抽象，一旦掌握了读图的方法后，会发现它们表现数据的能力还是很强的。

1. 茎叶图分析

茎叶图又称“枝叶图”，它的思路是将数组中的数按位数进行比较，然后将数的大小基本不变或变化不大的位作为一个主干，并将变化大的位的数作为分枝，列在主干的后面，这样就可以清楚地看到每个主干后面有几个数，每个数具体是多少。下面以产品单位成本数据为例，分析它的茎叶图，如表 6-2 所示。

表 6-2 产品单位成本数据

序号	产量(台)	单位成本(元/台)
1	4300	346.23

(续)

序号	产量(台)	单位成本(元/台)
2	4004	343.34
3	4300	327.46
4	5016	313.27
5	5511	310.75
6	5648	307.61
7	5876	314.56
8	6651	305.72
9	6024	310.82
...

在 R 语言中，使用 stem 函数进行茎叶图分析，其格式为：

```
> stem(变量, scale=长度, width=绘图宽度, atom=容差)
```

首先调用 read.table 方法读取数据文件，然后调用 stem 函数绘制茎叶图。代码如下：

```
>read.table("cp.csv",,header=TRUE,sep=",")->cp
> stem(cp$单机成本.元.台.,scale=2)
The decimal point is 1 digit(s) to the right of the |
```

stem 函数的 scale 参数为 2，表示将个数位分成两段，0~4 为一段，5~9 为另一段。
stem 函数生成的成本数据茎叶图如下：

```
29 | 68
30 | 1356778
31 | 1135
32 | 7
33 |
34 | 36
```

茎叶图的每一行表示每个茎与它的叶子，“|”前面是茎，而“|”后面是叶。以最后一行“34|36”为例，“|”前面的“34”是茎，后面的“36”是叶，这行的意思是：百位数为 3，十位数为 4 的数据有两个，分别是：345 和 346。

从茎叶图中可看出，单位成本主要集中在 300~309 元，因为代表茎 30 的行拥有的叶子最多。此外，茎 33 的行没有一个叶子，这说明没有一件产品的单位成本在 330~339 元这个范围内。

2. 箱线图分析

箱形图提供了一种只用 5 个点来对数据集做简单总结的方式，这 5 个点包括最大值、最小值、中位数、下四分位数和上四分位数。箱形图中最重要的内容是对相关统计点的计算，相关统计点可以通过百分位数计算方法进行实现。

以 2010 年全国就业调查数据为例，绘制“平均教育经费”的箱形图并进行分析。R 语言中，实现箱线图分析的相应函数为 boxplot。下面的代码绘制平均教育经费的箱形图。

```
> boxplot(jiuye$平均教育经费)
```

如图 6-4 所示的箱形图将平均教育经费很形象地分为中心、延伸以及分部状态的全部范围。中间那个箱子的顶部是上四分位数，底部是下四分位数，中间的粗线是中位数位置，箱体由上下伸出的垂直部分表示数据的散布范围。另外在散布范围外还有一些小圆点，那些是异常点，可见平均教育经费有一些特大值，最大的异常值超过了 12000。

除了箱形图外，在 R 语言中，还可以使用 `fivenum` 函数来分析前面说的 5 个点的概要。代码如下：

```
> fivenum(cp$单机成本.元.台.)
```

分析结果如下，它们分别是最小值、下四分位数、中位数、上四分位数、最大值。

```
[1] 296.210 304.275 307.225 313.915 346.230
```

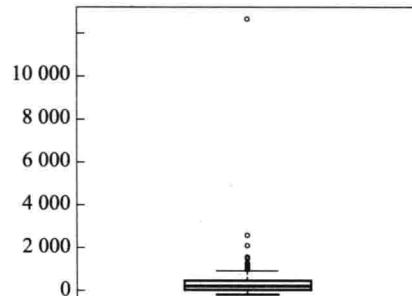


图 6-4 平均教育经费的箱形图

6.2 数据分布趋势案例解析

本节继续以产品成本和全国就业调查数据为例，剖析这两类数据的分布趋势。

6.2.1 平均值

下面使用 R 语言的 `mean` 函数统计就业调查数据中的“平均劳动报酬”。代码如下：

```
> mean(jiuye[["平均劳动报酬"]])
[1] 42365.36
```

同时，还可以统计一下“平均劳动报酬”和“平均教育经费”。代码如下：

```
> cbind(jiuye[["平均劳动报酬"]],jiuye[["平均教育经费"]])
> apply(jiuyeinfo,2,mean)
[1] 42365.365 391.035
```

6.2.2 加权平均值

加权平均数与算术平均数类似，但数据集中的每个数据对于平均数的贡献并不是相等的，有些数据要比其他的数据更加重要。因此，在加权平均法中，每个数据都有其相对应的权重。

以产品成本数据为例，剖析加权平均值。首先读取产品成本数据。代码如下：

```
> read.table("cp.csv",,header=TRUE,sep=",")->cp
> cp
   序号 产量.台. 单机成本.元.台.
1     1      4300        346.23
2     2      4004        343.34
3     3      4300        327.46
4     4      5016        313.27
```

5	5	5511	310.75
6	6	5648	307.61
7	7	5876	314.56
8	8	6651	305.72
9	9	6024	310.82
10	10	6194	306.83
11	11	7558	305.11
12	12	7381	300.71
13	13	6950	306.84
14	14	6471	303.44
15	15	6354	298.03
16	16	8000	296.21

然后，求产品平均单位成本。代码如下：

```
> weighted.mean(cp$单机成本.元.台., cp$产量.台.)
[1] 309.9866
```

6.2.3 数据排序

在 R 语言中，使用 `sort` 函数进行数据排序。比如，要对“平均教育经费”进行排序，可采用如下代码：

```
> sort(jiuye$平均教育经费)
[1]    0    0    0    2    2    2    6    7    7    8   10   13
[13]  27   30   31   31   31   32   35   37   38   42   42   44
[25]  46   50   51   55   55   62   63   65   66   66   67   71
[37]  72   72   75   75   76   80   89   92   93   93   95   95
[49] 100  100  100  100  100  105  109  110  111  115  118  119
[61] 125  136  138  143  144  145  146  147  147  149  149  157
[73] 159  161  161  162  162  166  168  168  168  172  177  177
[85] 182  184  186  188  190  196  196  196  200  201  206  210
[97] 210  212  212  221  224  225  230  230  241  241  247  247
[109] 258  260  267  267  276  276  277  282  295  295  298  299
[121] 303  304  305  306  308  314  315  317  330  332  337  340
[133] 341  342  348  367  369  371  374  374  389  389  396  402
[145] 405  409  416  422  422  423  431  436  443  454  455  461
[157] 466  470  486  502  522  524  535  551  551  554  555  557
[169] 563  571  582  645  679  682  692  722  738  753  768  782
[181] 818  830  832  840  840  858  890  890  890  986  995  1096
[193] 1131 1198 1255 1469 1553 2087 2564 12645
```

排序后，可以初步发现，这些行业的教育经费中，最大的有 12645，而最小的除 0 之外还有 2，不同行业之间的教育经费差异很大。

也可以改变排序顺序，通过指定 `decreasing` 参数为 `TRUE`，实现按从大到小的顺序排列。代码如下：

```
> sort(jiuye$平均教育经费, decreasing=TRUE)
[1] 12645  2564  2087  1553  1469  1255  1198  1131  1096  995  986  890
[13]  890   890   858   840   840   832   830   818   782   768   753   738
[25]  722   692   682   679   645   582   571   563   557   555   554   551
[37]  551   535   524   522   502   486   470   466   461   455   454   443
```

[49]	436	431	423	422	422	416	409	405	402	396	389	389
[61]	374	374	371	369	367	348	342	341	340	337	332	330
[73]	317	315	314	308	306	305	304	303	299	298	295	295
[85]	282	277	276	276	267	267	260	258	247	247	241	241
[97]	230	230	225	224	221	212	212	210	210	206	201	200
[109]	196	196	196	190	188	186	184	182	177	177	172	168
[121]	168	168	166	162	162	161	161	159	157	149	149	147
[133]	147	146	145	144	143	138	136	125	119	118	115	111
[145]	110	109	105	100	100	100	100	100	95	95	93	93
[157]	92	89	80	76	75	75	72	72	71	67	66	66
[169]	65	63	62	55	55	51	50	46	44	42	42	38
[181]	37	35	32	31	31	31	30	27	13	10	8	7
[193]	7	6	2	2	2	0	0	0				

>

6.2.4 中位数

中位数比平均值更有稳健性，因为它不受偏态分布的影响。代码如下：

```
> median(jiuye$平均教育经费) #中位数
[1] 222.5
> mean(jiuye$平均教育经费) #平均数
[1] 391.035
```

教育经费的中位数 222.5 与它的平均值 391.035 有一定差距，这说明平均教育经费不是对称分布的。

6.2.5 极差、半极差

极差是一组数据中最大数据与最小数据的差，用来刻画一组数据的离散程度，反映变量分布的变异范围和离散幅度，在样本总体中任何两个单位的标准值之差都不能超过极差。同时，它还能体现一组数据波动的范围。下面的代码计算“平均教育经费”的极差。

```
> max(jiuye$平均教育经费)-min(jiuye$平均教育经费)
[1] 12645
```

相对极差而言，四分位数间距（上四分位数与下四分位数之差）更稳定，它不受两端个别极大值或极小值的影响，可理解为中间 50% 观察值的极差，因此又被称为半极差。

四分位数是统计学中分位数的一种，即把所有数值由小到大排列并分成 4 等份，处于 3 个分隔点位置的得分就是四分位数，下四分位数是所有数据由小到大排列后处于 25% 位置的数，上四分位数是所有数据由小到大排列后处于 75% 位置的数。四分位数在 R 语言中用 quantile 函数求解，下面的代码计算了“平均教育经费”和“平均劳动报酬”的四分位数。

```
> quantile(jiuye$平均教育经费)
  0%    25%    50%    75%   100%
0.0   100.0  222.5  425.0 12645.0
> quantile(jiuye$平均劳动报酬)
  0%    25%    50%    75%   100%
13624.0 28607.5 37681.0 51762.0 150098.0
```

变异度反映数据围绕中心位的离散度，四分位数间距数值越大，变异度越大，反之，变异度越小。在 R 语言中使用 IQR 函数求解四分位数间距，下面的代码计算“平均教育经费”和“平均劳动报酬”的四分位数间距。

```
> IQR(jiuye$平均教育经费)
[1] 325
> IQR(jiuye$平均劳动报酬)
[1] 23154.5
```

从执行结果来看，“平均教育经费”相比“平均劳动报酬”变异度小很多。

6.2.6 方差

方差是重要的数据分散程度度量指标。其计算公式为：

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

在 R 语言中，可使用 var 函数统计方差。下面的代码计算“平均教育经费”的方差。

```
> var(jiuye$平均教育经费)
[1] 883263.6
```

6.2.7 标准差

标准差也是重要的数据分散程度度量指标。其计算公式为：

$$S = \sqrt{S^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

在 R 语言中使用 sd 函数统计标准差。下面的代码计算“平均教育经费”的标准差。

```
> sd(jiuye$平均教育经费)
[1] 939.821
```

6.2.8 变异系数、样本平方和

1. 变异系数

变异系数，又称“离散系数”，是概率分布离散程度的一个归一化量度，其定义为标准差与平均值之比。变异系数的计算公式为：

$$C.V = \frac{S}{\bar{x}} \times 100\%$$

上式中 S 表示标准差， \bar{x} 表示平均值。变异系数越小，变异程度越小；反之，变异系数越大，变异程度越大。下面的代码计算了“平均教育经费”的变异系数。

```
> sd(jiuye$平均教育经费) / mean(jiuye$平均教育经费)
[1] 2.403419
```

再看看“平均劳动报酬”的变异系数。代码如下：

```
> sd(jiuye$平均劳动报酬) / mean(jiuye$平均劳动报酬)
```

```
[1] 0.4916487
```

可见，“平均教育经费”相对于“平均劳动报酬”分布更分散，因为它的变异系数更高。

2. 样本平方和

样本校正平方和（CSS）为样本与均值差的平方求和。下面的代码计算“平均教育经费”的样本校正平方和。

```
> sum((jiuye$平均教育经费-mean(jiuye$平均教育经费))^2)
[1] 175769451
```

样本未校正平方和（USS）为样本值平方的求和。下面的代码计算了“平均教育经费”的样本未校正平方和。

```
> sum(jiuye$平均教育经费^2)
[1] 206351125
```

6.2.9 偏度系数、峰度系数

1. 偏度系数

在统计学中，偏度系数是用于衡量实数随机变量概率分布的不对称性的。偏度的值可以为正，可以为负，是无量纲的量，其取值通常为-3~+3，其绝对值越大，表明偏斜程度越大。均值右侧更分散的数据偏度系数为正，左侧更分散的数据偏度系数为负。

偏度系数的计算公式为：

$$\gamma = \frac{n}{(n-1)(n-2)S^3} \sum (x_i - \bar{x})^3$$

在 R 语言中，可用如下代码计算“平均教育经费”的偏度系数。

```
> mean(jiuye$平均教育经费) -> mymean
> sd(jiuye$平均教育经费) -> mysd
> length(jiuye$平均教育经费) -> myn
> jiuye$平均教育经费 -> x
> myn / ((myn-1) * (myn-2)) * sum((x-mymean)^3) / mysd^3
[1] 11.36649
```

2. 峰度系数

峰度系数衡量实数随机变量概率分布的峰态，峰度高就意味着方差增大是由低频度的大于或小于平均值的极端差值引起的。当数据分布为正态分布时，峰度系数近似为0；当数据分布较正态分布的尾部更分散时，峰度系数为正，两侧的极端数据较多；除此以外，峰度系数为负，两侧的极端数据较少。

峰度系数计算公式为：

$$K = \frac{n(n+1)}{(n-1)(n-2)(n-3)S^4} \sum (x_i - \bar{x})^4 - \frac{(n-1)^2}{(n-2)(n-3)}$$

在 R 语言中，可用如下代码计算“平均教育经费”的峰度系数。

```
> mean(jiuye$平均教育经费) -> mymean
```

```

> sd(jiuye$平均教育经费)->mysd
> length(jiuye$平均教育经费)->myn
> jiuye$平均教育经费->x
> ((myn*(myn+1)) / ((myn-1)*(myn-2)*(myn-3)) * sum((x-mymean)^4) / mysd^4 - (3*(myn-
1)^2) / ((myn-2)*(myn-3)))
[1] 146.8809

```

6.3 正态分布案例解析

6.3.1 正态分布函数

对于一维实随机变量 X , 设它的累积分布函数是 $F_X(x)$ 。如果存在可测函数 $f_X(x)$, 满足:

$$\forall -\infty < a < \infty, F_X(a) = \int_{-\infty}^a f_X(x) dx$$

那么 X 是一个连续型随机变量, 并且 $f_X(x)$ 是它的概率密度函数。

累积分布函数, 又叫累计分布函数, 是概率密度函数的积分, 能完整地描述一个实随机变量 X 的概率分布情况。对于所有实数 x , 累积分布函数的定义如下:

$$F(x) = P(X \leq x)$$

正态分布的累积分布函数为:

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x \exp\left\{-\frac{(t-\mu)^2}{2\sigma^2}\right\} dt$$

其中, μ 是均值, σ 是方差。

正态分布的概率密度曲线通常如图 6-5 所示。

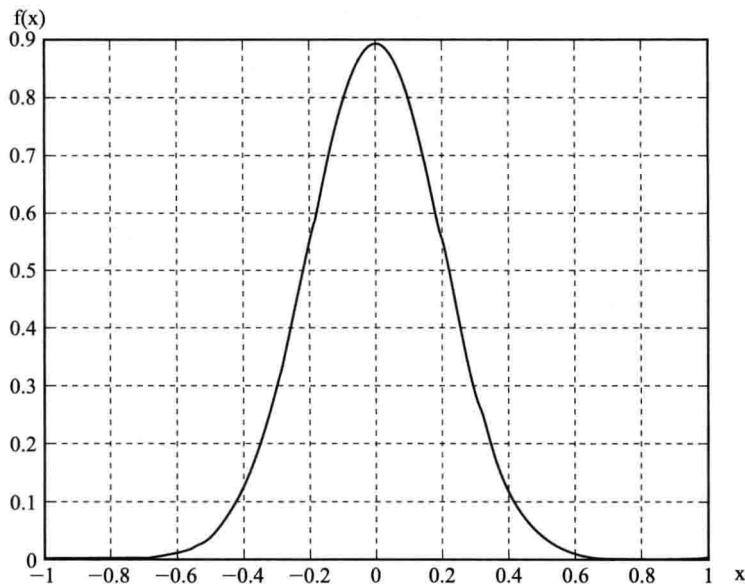


图 6-5 正态分布的概率密度曲线

说到正态分布，不得不提一下偏态分布，偏态分布是指频数分布不对称，集中位置偏向于一侧，若集中位置偏向数值小的一侧，则称为正偏态分布；如果集中位置偏向数值大的一侧，则称为负偏态。如图 6-6 所示，左边为负偏态，右边为正偏态。

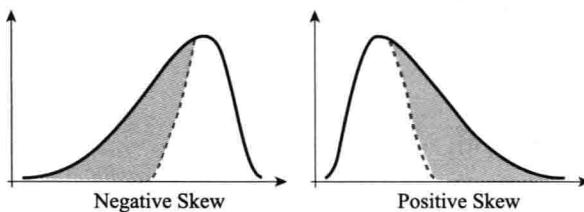


图 6-6 偏态分布

6.3.2 峰度系数分析

可用峰度系数计算“平均劳动报酬”相对于“平均教育经费”哪个更接近正态分布。代码如下：

```
> mean(jiuye$平均劳动报酬) ->mymean
> sd(jiuye$平均劳动报酬) ->mysd
> length(jiuye$平均劳动报酬) ->myn
> jiuye$平均劳动报酬->x
> ((myn * (myn+1)) / ((myn-1) * (myn-2) * (myn-3)) * sum((x-mymean)^4) / mysd^4 - (3 * (myn-1)^2) / ((myn-2) * (myn-3)))
[1] 5.417817
```

上面计算出了“平均劳动报酬”的峰度系数为 5.417817，“平均教育经费”的峰度系数为 146.8809（见 6.2.9 节）。这两个峰度系数表明，“平均劳动报酬”相对于“平均教育经费”更接近正态分布。

从下面的分析中可以发现，产品产量最适合正态分布模型，因为它的峰度系数仅为 -0.6830728，非常接近正态分布。

```
> mean(cp$产量.台.) ->mymean
> sd(cp$产量.台.) ->mysd
> length(cp$产量.台.) ->myn
> cp$产量.台.->x
> ((myn * (myn+1)) / ((myn-1) * (myn-2) * (myn-3)) * sum((x-mymean)^4) / mysd^4 - (3 * (myn-1)^2) / ((myn-2) * (myn-3)))
[1] -0.6830728
```

6.3.3 累积分布概率

在使用 `pnorm` 求产品产量的分布函数时，对应的每个实数随机变量都有其累积分布概率。下面的代码计算产品产量的累积分布概率。

```
> mean(cp$产量.台.) ->mymean
> sd(cp$产量.台.) ->mysd
```

```

> length(cp$产量.台.)->myn
> cp$产量.台.->x
>x
[1] 4300 4004 4300 5016 5511 5648 5876
6651 6024 6194 7558 7381 6950 6471
[15] 6354 8000
> pnorm(x,mymean,mysd)
[1] 0.07435941 0.04519643 0.07435941
0.20013522 0.33567136 0.37868351
[7] 0.45345196 0.70390728 0.50306546
0.55994848 0.90310411 0.87500925
[13] 0.78449233 0.64954647 0.61239714
0.95270286

```

为了更好地观察效果，再绘制一张产品产量的累积分布概率的散点图。

```
> plot(x, pnorm(x, mymean, mysd))
```

绘制结果如图 6-7 所示。

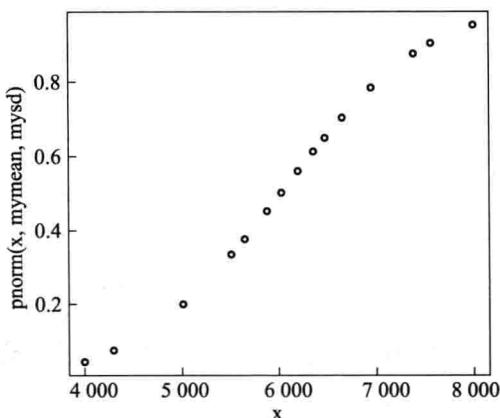


图 6-7 产品产量累积分布

6.3.4 概率密度函数

1. 概率密度概述

一个连续型随机变量的概率密度函数（简称为密度函数）是描述这个随机变量的输出值在某一个确定的取值点附近的可能性的函数。随机变量的取值落在某个区域之内的概率则是概率密度函数在这个区域上的积分，当概率密度函数存在的时候，累积分布函数则是概率密度函数的积分。

正态分布的概率密度函数为：

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(t-\mu)^2}{2\sigma^2}\right\}$$

其中， μ 是均值， σ 是方差。

其概率密度曲线关于 $x=\mu$ 对称。

2. 概率密度函数计算

在 R 语言中，使用 `dnorm(变量, 平均值, 标准差)` 求解概率密度函数。下面的代码计算产品产量的概率密度。

```

> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> length(cp$产量.台.)->myn
> cp$产量.台.->x
> dnorm(x, mymean, mysd)
[1] 1.184240e-04 8.009886e-05 1.184240e-04 2.358477e-04 3.070239e-04
[6] 3.202882e-04 3.336542e-04 2.910430e-04 3.359337e-04 3.321435e-04
[11] 1.444115e-04 1.733374e-04 2.463862e-04 3.120546e-04 3.225207e-04
[16] 8.307503e-05

```

查看产品产量均值，得到如下结果：

```
> mymean  
[1] 6014.875
```

绘制散点图如图 6-8 所示。可以看到该曲线接近于以 6014.875 为对称点的对称分布。
绘制代码如下：

```
> plot(x,dnorm(x,mymean,mysd))
```

此外，`rnorm` 还可以返回正态分布随机数，调用格式为 `rnorm(长度 , 平均值 , 标准差)`。
比如：

```
>rnorm(50,0,1)->rx  
> plot(rx,dnorm(rx))
```

如图 6-9 所示是一个经典的正态密度曲线，从曲线上看就像一个驼峰。

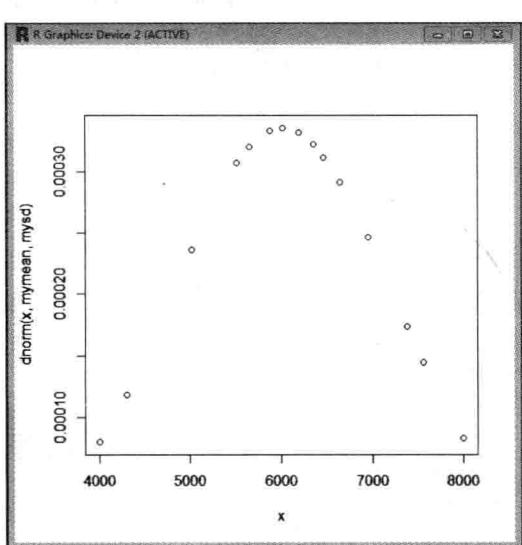


图 6-8 产品产量概率密度图

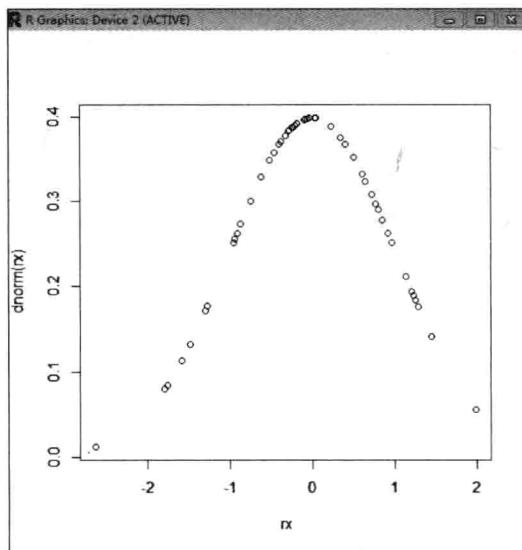


图 6-9 正态密度曲线

6.3.5 分位点

分位点分为上 α 分位点与下 α 分位点。

1. 下 α 分位点

可从概率密度函数的角度理解下 α 分位点。设连续随机变量 X 的累积分布函数为 $F(x)$ ，密度函数为 $f(x)$ ，则有：

$$F(x) = \int_{-\infty}^x f(x) dx = p(X \leq Z_\alpha) = \alpha$$

上式的含义为：连续随机变量 X 小于等于 Z_α 的概率为 α 。在这里，称 Z_α 是 X 的下 α 分

位点。

下面计算产量分布的下 α 分位点。设下 α 分位点的 $\alpha=25\%=0.25$ ，需要分析产量小于多少的概率为 25%。

计算产量分布的下 α 分位点 ($\alpha=25\%$)，可如下调用 qnorm 函数（其中，mean 为平均值，sd 为标准差）：

```
qnorm(0.25, mean, sd)
```

具体代码如下：

```
> mean(cp$产量.台.)->mymean  
>sd(cp$产量.台.)->mysd  
>qnorm(0.25, mean=mymean, sd=mysd)  
[1] 5213.9
```

上面的计算结果表明，产量 <5213.9 的概率为 25%。

2. 上 α 分位点

上 α 分位点的公式为：

$$F(x)=\int_a^{+\infty} f(x)dx=p(x>Z_\alpha)=1-p(x\leq Z_\alpha)=\alpha$$

其中， Z_α 为 X 的上 α 分位点。

分析上 α 分位点的公式可发现，上 α 分位点的计算与下 α 分位点有关，比如：计算上 α 分位点 ($\alpha=25\%$) 可转化为计算下 α 分位点 ($\alpha=1-25\%=75\%$)。

下面计算产量分布的上 α 分位点。设上 α 分位点的 $\alpha=25\%$ ，需要分析产量大于多少的概率为 25%。

计算产量分布的上 α 分位点 ($\alpha=25\%$)，可如下调用 qnorm 函数（其中，mean 为平均值，sd 为标准差）：

```
qnorm(1-0.25, mean, sd)
```

具体代码如下：

```
> mean(cp$产量.台.)->mymean  
>sd(cp$产量.台.)->mysd  
>qnorm(0.75, mean=mymean, sd=mysd)  
[1] 6815.85
```

上面的计算结果表明，产量 >6815.85 的概率为 25%，即上 α 分位点的公式中， Z_α 为产量 6815.85，上 α 分位点的 α 为 0.25，可用下式表示：

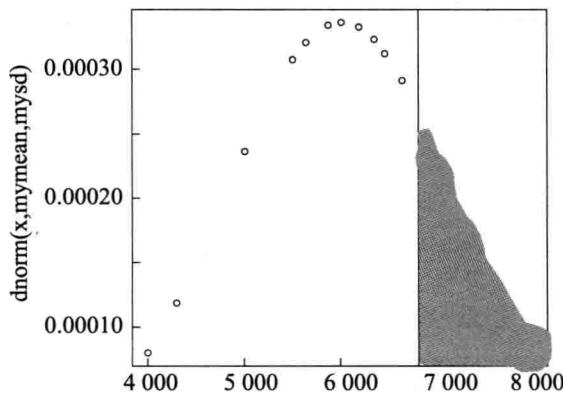
$$P(x>Z_\alpha)=p(x>6815.85)=0.25$$

3. 绘效果图

(1) 通过下面 R 语句绘制如图 6-10 所示的产品产量的概率密度图， $P(x>0.25)$ 为图 6-10 中的阴影面积（根据积分的几何意义，累积分布函数 $F(x)$ 是密度函数 $f(x)$ 的积分，图中若干点组成了密度函数曲线，而曲线与 X 轴围成的面积则为累积分布）。

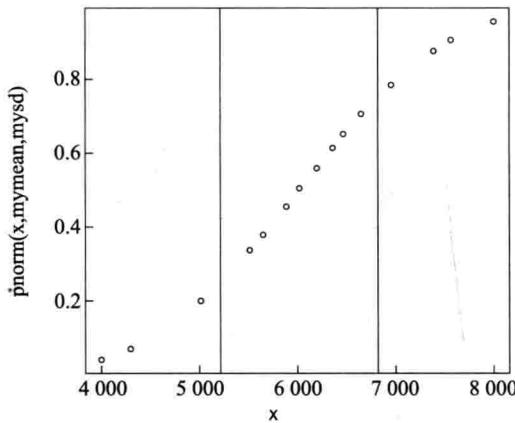
```
> mean(cp$产量.台.)->mymean
```

```
>sd(cp$产量.台.)->mysd
>cp$产量.台.->x
>plot(x,dnorm(x,mymean,mysd))
>abline(v=6815.85)
```

图 6-10 产品产量的上 α 分位点

(2) 绘制产品产量的累积分布图(如图 6-11 所示)。可绘制一个产品产量的上 $\alpha(\alpha=0.25)$ 分位点和下 $\alpha(\alpha=0.25)$ 分位点的效果图。绘制代码如下:

```
> mean(cp$产量.台.)->mymean
>sd(cp$产量.台.)->mysd
> cp$产量.台.->x
>plot(x,pnorm(x,mymean,mysd))
>abline(v=6815.85)
>abline(v=5213.85)
```

图 6-11 产量的上 α 分位点和下 α 分位点

分析绘制图 6-10 与图 6-11 所示的结果,能较直观地验证刚才得到的结论: 上 $\alpha(\alpha=0.25)$ 分位点表明产量 >6815.85 的概率为 25%,而下 $\alpha(\alpha=0.25)$ 分位点表明产量 <5213.9 的概率为 25%。

6.3.6 频率直方图

在 R 语言中，可使用 hist 语句（设 freq 参数为 TRUE）生成频率直方图。下面的代码绘制“平均劳动报酬”的频率直方图。

```
> hist(jiuye[["平均劳动报酬"]], freq=TRUE)
```

绘制结果如图 6-12 所示。

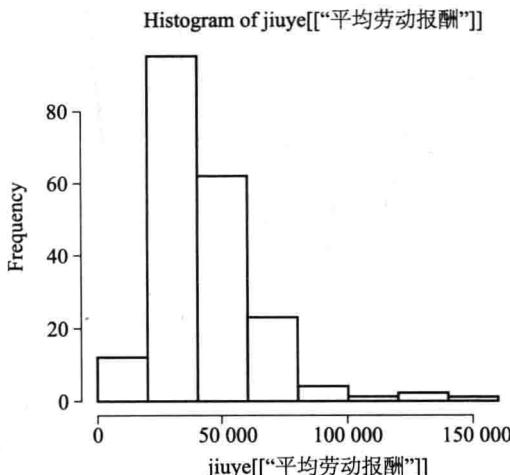


图 6-12 平均劳动报酬的频率直方图

6.3.7 核概率密度与正态概率分布图

1. 核概率密度与正态概率

下面考虑让“平均劳动报酬”的概率密度与正态分布在一张图中显示出来，这样就能更好地看清数据的分布情况。示例代码如下：

```
> hist(jiuye[["平均劳动报酬"]], freq=FALSE)
> lines(density(jiuye[["平均劳动报酬"]]), col="red")
> x<-c(0:ceiling(max(jiuye[["平均劳动报酬"]])))
>
lines(x,dnorm(x,mean(jiuye[["平均劳动报酬"]]),sd(jiuye[["平均劳动报酬"]])),col="blue")
```

绘制结果如图 6-13 所示。

从图 6-13 中可以看到，“平均劳

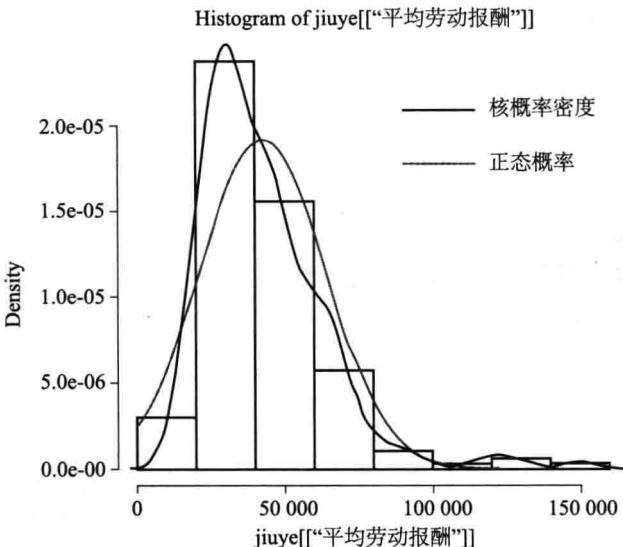


图 6-13 平均劳动报酬的概率密度与正态分布

动报酬”的偏度大于 0，直方图偏左，属于偏态分布。

2. 经验累积分布与正态分布

经验分布函数是指根据样本构造的概率分布函数。设 x_1, x_2, \dots, x_n 为一组样本，定义函数 $m(x)$ 表示样本中小于或者等于 x 的样本个数，则称函数

$$F_n^x = \frac{m(x)}{n}$$

为样本 x_1, x_2, \dots, x_n 的经验分布函数。

下面的代码绘制“平均劳动报酬”的经验累积分布与正态分布。

```
> plot(ecdf(jiuye[["平均劳动报酬"]]), verticals=TRUE, do.p=FALSE)
> lines(x, pnorm(x, mean(jiuye[["平均劳动报酬"]]), sd(jiuye[["平均劳动报酬"]])), col="blue")
```

绘制结果如图 6-14 所示。其中，光滑的线为累积正态分布曲线，不光滑的线为经验累积分布曲线。

6.3.8 正太检验与分布拟合

1. QQ 图

QQ 图可以测试数据分布是否近似为某种类型分布。如果近似于正态分布，则数据点接近下面方程表示的直线。

$$y = \sigma x + \mu$$

其中， σ 为标准差， μ 为平均数。

比如，可用它来测试“平均劳动报酬”分布是否近似于正态分布。在 R 语言中，使用 `qqnorm` 函数来画数据点图，使用 `qqline` 函数画这根直线，如图 6-15 所示。代码如下：

```
> qqnorm([["平均劳动报酬"]])
> qqline(jiuye[["平均劳动报酬"]])
```

从图 6-15 来看，平均劳动报酬离标准正态分布还是有差距的。

相对于“平均劳动报酬”，产品产量就非常接近正态分布。代码如下：

```
> qqnorm(cp$产量.台.)
```

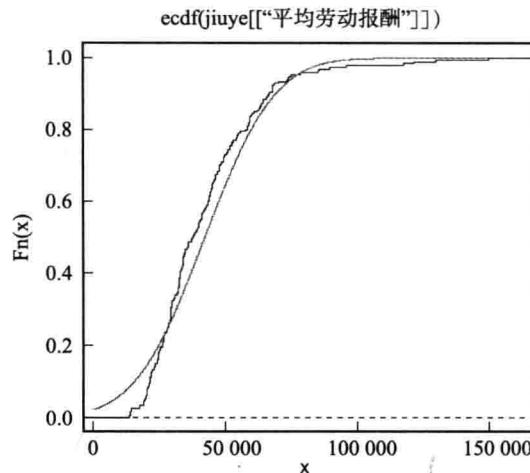


图 6-14 平均劳动报酬的经验累积分布与正态分布

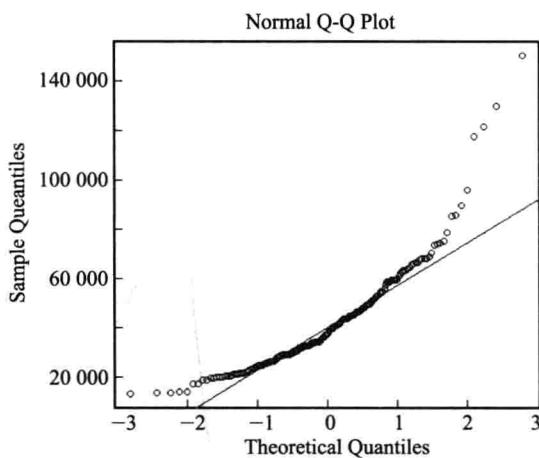


图 6-15 平均劳动报酬分布 QQ 图

```
> qqline(cp$产量.台.)
```

绘制出的 QQ 图如图 6-16 所示。

2. 正态检验与分布拟合

(1) W 检验。W 检验可以检验数据是否符合正态分布。在 R 语言中使用函数 shapiro.test() 进行正态 W 检验。代码如下：

```
> shapiro.test(cp$产量.台.)
Shapiro-Wilk normality test
data: cp$产量.台.
W = 0.9671, p-value = 0.7903
```

上述代码中出现了 p 值，其作用是：当 p 值小于某个显著水平 α （如 0.05）时，认为样本不是来自于正态分布的总体。此例中， $0.7903 > 0.05$ ，可认为产量是正态分布的。

(2) Kolmogorov-Smirnov 检验。Kolmogorov-Smirnov 检验用于检验单一样本是否来自某一特定分布，比如检验一组数据是否为正态分布。它的检验方法是以样本数据的累计频数分布与特定理论分布做比较，若两者间的差距很小，则该样本取自某特定分布族。可比较一个频率分布 $f(x)$ 与理论分布 $g(x)$ ，或者两个观测值分布来完成检验。

可以从假设检验的角度来理解 Kolmogorov-Smirnov 检验，假设检验使用了一种类似于“反证法”的推理方法，它先假设总体中的某项假设成立，计算其会导致什么结果产生。若导致不合理现象发生，拒绝原先的假设；若没有导致不合理的现象发生，即不拒绝原假设，从而接受原假设。

对 Kolmogorov-Smirnov 检验而言，将原假设 H_0 确定为：两个数据分布一致或者数据符合特定理论分布。用 $F_0(x)$ 表示分布函数， $F_n(x)$ 表示一组随机样本的累积概率函数，设 D 为 $F_0(x)$ 与 $F_n(x)$ 差距的最大值，定义如下：

$$D = \max |F_n(x) - F_0(x)|$$

当实际观测值 $D > D(n, \alpha)$ 时，则拒绝 H_0 ，否则接受 H_0 假设。

在 R 语言中使用 ks.test 函数完成正态性检验。代码如下：

```
ks.test(x, y, ..., alternative = c("two.sided", "less", "greater"),
exact = NULL)
```

上述代码中有 4 个参数，第一个参数 x 为观测值向量；第二个参数 y 为第二观测值向量或者累计分布函数，或者一个真正的累积分布函数，如 pnorm，只对连续 CDF 有效；第三个参数指明是单侧检验还是双侧检验；exact 参数为 NULL 或者一个逻辑值，表明是否需要计算精确的 P 值。比如：要生成两个随机的正态分布，然后检验这两个分布是否是同一类

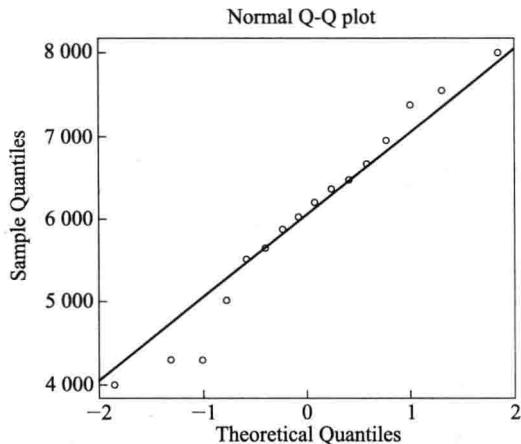


图 6-16 产品产量 QQ 图

型的分类。其示例代码如下：

```
> ks.test(rnorm(80), rnorm(40))
Two-sample Kolmogorov-Smirnov test
data: rnorm(80) and rnorm(40)
D = 0.125, p-value = 0.7874
alternative hypothesis: two-sided
```

从上述代码可见， p 值大于 0.05，不拒绝原假设，因此可认为这两个分布是同一类型。

Kolmogorov-Smirnov 检验要求待验分布是连续的，连续分布出现相同值的概率为 0，也就是说，数据中若出现相同值，则连续分布的假设不成立。

6.3.9 其他分布及其拟合

前面几节介绍了 R 语言函数对正态分布的支持（函数名除前缀外为 norm）。除了正态分布外，R 语言还支持对其他数据分布的分析和拟合，如下所示：

指数分布	<code>rexp(n, rate=1)</code>
gama 分布	<code>rgamma(n, shape, scale=1)</code>
泊松分布	<code>rpois(n, lambda)</code>
Weibull 分布	<code>rweibull(n, shape, scale=1)</code>
Cauchy 分布	<code>rcauchy(n, location=0, scale=1)</code>
beta 分布	<code>rbeta(n, shape1, shape2)</code>
S(tudent) 分布	<code>rt(n, df)</code>
Fisher-Snedecor	<code>rf(n, df1, df2)</code>
Pearson	<code>rchisq(n, df)</code>
二项式分布	<code>rbinom(n, size, prob)</code>
多项式分布	<code>rmultinom(n, size, prob)</code>
几何分布	<code>rgeom(n, prob)</code>
hypergeometric	<code>rhyper(nn, m, n, k)</code>
logistic	<code>rlogis(n, location=0, scale=1)</code>
lognormal	<code>rlnorm(n, meanlog=0, sdlog=1)</code>
negative binomial	<code>rnbinom(n, size, prob)</code>
uniform	<code>runif(n, min=0, max=1)</code>
Wilcoxon's statistics	<code>rwilcox(nn, m, n), rsignrank(nn, n)</code>

对这些函数的调用格式是：前缀 + 函数名。

前缀规则如下：

密度函数：d

累计概率分布函数：p

分布函数的反函数：q

相同分布的随机数：r

6.4 小结

对于统计分析来说，最好的学习方式就是实践，光看理论比较难于理解。本章以实例的形式分析了正态分布函数、峰度系数、累积分布概率、概率密度函数及曲线、分位点、正太检验与分布拟合等数据统计指标。在数据分布模型中，除了正态分布，还有很多分布模型，在本章结尾处概述了 R 语言提供的其他分布分析函数。本章介绍的数据分布分析与第 5 章介绍的回归分析构成了数据分析的基础，也是数据分析中常用的方法。

思考题

- (1) 下载本书例子中的美国地震台数据 `earthquakes.csv`，对美国地震台网数据中震深 `Depth` 进行分析，绘制累积分布概率的散点图。
- (2) 下载本书例子中的美国地震台数据 `earthquakes.csv`，对美国地震台网数据中震级 `Magnitude` 进行分析，分析概率密度。
- (3) 下载本书例子中的全国就业调查情况数据 `youxiangz.csv`，分析“平均劳动报酬”的正态分布函数、峰度系数、累积分布概率、概率密度函数及曲线、频率直方图。

第四部分

机器学习实战篇

自古圣贤之言学也，咸以躬行实践
为先，识见言论次之。

——林希元

第7章

机器学习算法

学习就是在不断重复的工作中使自己的能力不断增强或改进，在下一次执行相同或类似的任务时，会比原来做得更好或效率更高。机器学习研究指的是用计算机来模拟或实现人类学习活动，研究如何使机器通过识别和利用现有的知识来获取新知识和新技能。机器学习的内部表现为从未知到已知这样一个知识增长过程，其外部表现为某些性能和适应性得到系统的改善。机器学习形成了一套算法理论，这些算法使系统能完成原来不能完成的任务，或者能更好地完成原来可以完成的任务。

7.1 神经网络

1. 神经网络基本原理

人脑由上千亿条神经组成，每条神经平均又会连接到几千条其他的神经，通过这种连接方式，神经可以收发不同数量的能量。神经一个非常重要的功能就是，它们对能量的接收并不是立即作出响应，而是将它们累加起来，当这个累加的总和达到某个临界阈值时，它们才将自己的那部分能量发送给其他的神经。大脑通过调节这些连接的数目和强度来进行学习。

如图 7-1 所示是一个神经元的模型，神经网络复杂多样，由大量与该图类似的神经元及突触组成。神经科学界的共识是，人类大脑包含 1000 亿个神经元，每个神经元有 1 万个突触，数量巨大，组合方式复杂，联系广泛。也就是说，突触传递的机制复杂。

现在已经发现和阐明的突触传递机制有：突触后兴奋、突触后抑制、突触前抑制、突触前兴奋，以及远程抑制等。在突触传递机制中，释放神经递质是实现突触传递机能的中心环节，而不同的神经递质有着不同的作用性质和特点。

人工神经网络（ANN）是一种模仿生物神经网络结构和功能的数学模型，它使用大量的人工神经元连接来进行计算，该网络由大量的“神经元”相互连接构成，每个“神经元”代表一种特定的输出函数。又称为激励函数。每两个“神经元”间的连接代表一个通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则根据网络的连

接规则来确定，输出因权重值和激励函数的不同而不同。人工神经网络可理解为对自然界某种算法或者函数的逼近。

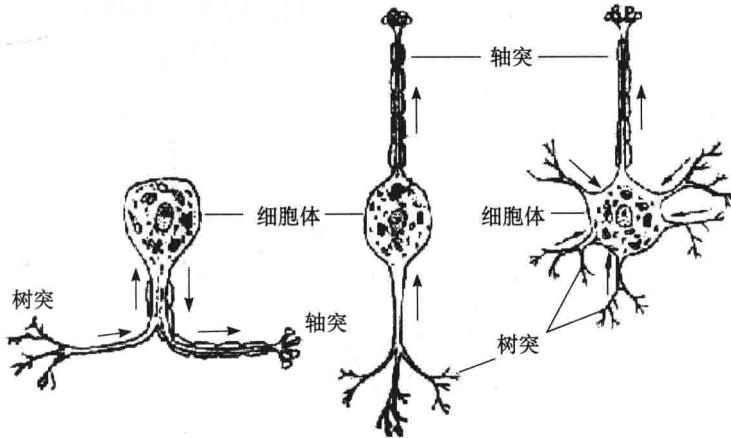


图 7-1 大脑神经元

如图 7-2 所示是一个简单的人工神经元示意图。其中， $x_i(t)$ 等数据为这个神经元的输入，代表其他神经元或外界对该神经元的输入； w_{ij} 等数据为这个神经元的权重， $u_i = \sum_j w_{ij} \cdot x_j(t)$ 是对输入的求和，为诱导局部域； $y_i = f(u_i(t))$ 为输出函数，也称激励函数，是对诱导局部域的再加工，也是最终的输出。

2. 神经网络发展历史

20 世纪 40 年代，心理学家 McCulloch 和数学家 Pitts 联合提出了兴奋与抑制型神经元模型，另外，Hebb 提出了神经元连接强度的修改规则；到 20 世纪五六十年代，该领域具有代表性的工作是 Rosenblatt 的感知机和 Widrow 的自适应性元件 Adaline；1969 年，Minsky 和 Papert 合作出版了颇有影响力的著作《感知器》，书中暗示感知器具有严重局限，得出了消极悲观的论点，使感知器与连接主义遭到冷落，而感知器是神经网络的一种重要形式。在 20 世纪 70 年代，人工神经网络的研究处于低潮，20 世纪 70 年代末，传统的冯·诺依曼（Von Neumann）数字计算机在模拟视听觉的人工智能方面遇到了物理上不可逾越的极限。但与此同时，Rumelhart、McClelland 和 Hopfield 等人在神经网络领域取得了突破性进展，神经网络的热潮再次掀起。

7.1.1 Rosenblatt 感知器

Rosenblatt 感知器是由美国计算机科学家罗森布拉特（F.Rosenblatt）于 1957 年提出的。F.Rosenblatt 经过证明得出结论，如果两类模式是线性可分的（指存在一个超平面将它们分

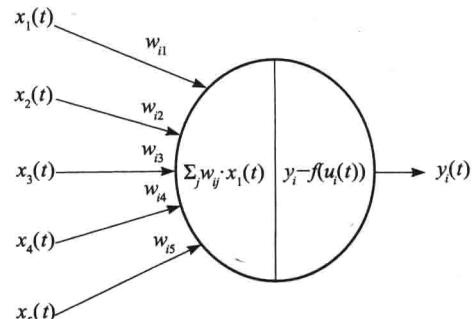


图 7-2 人工神经元

开), 则算法一定收敛。Rosenblatt 感知器特别适用于简单的模式分类问题, 也可用于基于模式分类的学习控制。

Rosenblatt 感知器建立在一个线性神经元之上, 神经元模型的求和节点计算作用于突触输入的线性组合, 同时结合外部作用的偏置, 对若干个突触的输入项求和后进行调节。

1. 基本计算过程

Rosenblatt 感知器的基本计算步骤如下:

- (1) 将数据作为输入送入神经元。
- (2) 通过权值和输入共同计算诱导局部域, 诱导局部域是指求和节点计算得到的结果, 计算公式如下:

$$v = \sum_{i=1}^m w_i x_i + b$$

(3) 以硬限幅器为输出函数, 诱导局部域被送入硬限幅器, 形成最终的输出硬限幅器的工作原理如下。

硬限幅器输入为正时, 神经元输出 +1, 反之输出 -1。计算公式为:

$$f(v) = \begin{cases} 1, & v \geq 0 \\ -1, & v < 0 \end{cases}$$

硬限幅器函数图像如图 7-3 所示。

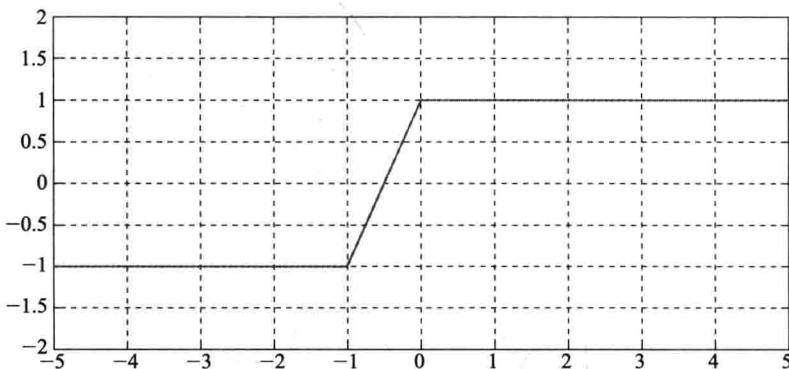


图 7-3 硬限幅器函数图像

从图 7-3 可以看出, 在最终的计算结果中, 把外部的输入 x_1, x_2, \dots, x_m 输出为两类, 分类规则是: 如果输出函数是 +1, 则为类 1; 如果输出为 -1, 则为类 2。感知器被超平面分为两类, 这个超平面为:

$$\sum_{i=1}^m w_i x_i + b = 0$$

2. 权值修正

在上述计算过程中, 第二步涉及一个重要的参数, 即权值。如何确定权值才能保证输出能被正确地分类到 +1 和 -1 呢?

首先，会产生一个初始权值，由初始权值计算得到的输出结果肯定会有误差。接着，要想办法让误差减少，这个过程就是权值 w 修正的过程。

新的问题产生了，哪些数据用来输入呢？在用神经网络进行机器学习时，输入数据是确定的，但输出结果是未知的，这些输出结果正是我们需要用神经网络预测的结果。解决这个问题的关键在于样本。样本是研究中实际观测或调查的一部分，研究对象的全部称为总体，样本必须能够正确反映总体情况。所以，首先要用样本将神经网络训练好，在确定权值后，再用训练好的神经网络对未知输入所属的输出进行预测。权值修正方法分为单样本修正算法和批量修正算法。

单样本修正算法的步骤为：神经网络每次读入一个样本，进行修正，样本读取完毕，修正过程结束。算法过程描述如下：

(1) 设置如下参数：

$$\begin{aligned} w(0) &= 0 \\ x(n) &= (+1, x_1(n), \dots, x_n(n))^T \\ w(n) &= (b, w_1(n), \dots, w_m(n))^T \end{aligned}$$

其中， b 为偏置， x 为输入向量， w 为权值。

(2) 感知器激活。

对于每个时间步 n ，通过输入向量 $x(n)$ 和期望输出 $d(n)$ 激活感知器。

(3) 计算感知器的输出。

$$\begin{aligned} y(n) &= \text{sgn}(w^T(n)x(n)) \\ \text{sgn} &= \begin{cases} +1, & v \geq 0 \\ -1, & v < 0 \end{cases} \end{aligned}$$

其中， n 为时间步， $x(n)$ 为输入向量， $w(n)$ 为权值向量， sgn 为硬限幅函数， v 为硬限幅函数的输入值。

(4) 更新感知器的权值向量。

$$\begin{aligned} w(n+1) &= w(n) + \eta(d(n) - y(n))x(n) \\ 0 < \eta &< 1 \\ d(n) &= \begin{cases} +1, & x(n) \text{ 属于第 1 类} \\ -1, & x(n) \text{ 属于第 2 类} \end{cases} \end{aligned}$$

其中， n 为学习速率（调整更新的步伐）。

下面用神经网络学习逻辑与的运算，用 Python 实现上述算法。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-1.py
import numpy as np
b=0
a=0.5
x = np.array([[b,1,1],[b,1,0],[b,0,0],[b,0,1]])
```

```

d =np.array([1,1,0,1])
w=np.array([b,0,0])
def sgn(v):
    if v>=0:
        return 1
    else:
        return 0
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    return oldw+a*(myd-comy(oldw,myx))*myx
i=0
for xn in x:
    w=neww(w,d[i],xn,a)
    i+=1
for xn in x:
    print "%d and %d => %d "%(xn[1],xn[2],comy(w,xn))

```

如下程序输出结果正确。

```

1 and 1 => 1
1 and 0 => 1
0 and 0 => 0
0 and 1 => 1

```

现在来测试一个更复杂的例子。针对下面两类函数训练神经网络，将一组 (x, y) 值划分为以下两类函数之一：

- $2x+1=y$ 为第 1 类。
- $7x+1=y$ 为第 2 类。

代码如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#7-2.py
import numpy as np
b=1
a=0.3
x=np.array([[b,1,3],[b,2,3],[b,1,8],[b,2,15],[b,3,7],[b,4,29]])
d=np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
def sgn(v):
    if v>=0:
        return 1
    else:
        return -1
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    print comy(oldw,myx)
    return oldw+a*(myd-comy(oldw,myx))*myx
i=0

```

```

for xn in x:
    print xn
    w=neww(w,d[i],xn,a)
    i+=1
    print w
for xn in x:
    print "%d %d => %d "%(xn[1],xn[2],comy(w,xn))
test=np.array([b,9,19])
print "%d %d => %d "%(test[1],test[2],comy(w,test))
test=np.array([b,9,64])
print "%d %d => %d "%(test[1],test[2],comy(w,test))

```

在上面代码中，使用了 [1,3]、[2,3]、[1,8]、[2,15]、[3,7]、[4,29] 这几组数据对该神经网络进行训练。运行该程序，对从未在样本中出现过的数据 [9,19]、[9,64] 进行分类。通过前面的函数定义可以知道，[9,19] 属于第 1 类，[9,64] 属于第 2 类。如下所示：

```

9 19 => 1
9 64 => -1

```

接着输出训练之后的神经网络权值参数，代码如下：

```

>>> w
array([ 1.,  1.2, -0.6])

```

根据上述参数，可以确定神经网络的分类线方程为：

$$1.2x - 0.6y + 1 = 0 \Rightarrow 1.2x + 1 = 0.6y \Rightarrow y \approx 2x + 1.68$$

那么对于不完全符合上述两个函数定义的数据，通过训练好的神经网络也能近似地划分到上述函数中。试着将前面的代码修改一下，加入两个不太规则的测试数据点 [9,16]、[9,60]，同时加入绘图命令，绘制数据点，然后使用上面计算的分类线方程绘制分类线（这样能直观地观察结果），如图 7-4 所示。

在如图 7-4 所示的样本数据点中，大的圆点属于第 2 类，输出为 -1，星号属于第 1 类，输出为 1；在测试数据点中，叉号属于第 2 类，输出为 -1，小点属于第 1 类，输出为 1。中间的虚线就是分类线，这条分类线把坐标系内的点分成为两类，分类线以上的点输出为 -1，以下的点输出为 1，训练好的神经网络通过这条分类线决定如何对输入所属的分类进行预测。

修改后的 Python 代码如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#7-3.py
import numpy as np
import pylab as pl
b=1
a=0.3

```

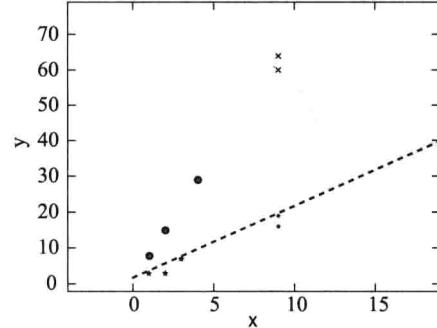


图 7-4 神经网络分类（附彩图）

```

x=np.array([[b,1,3],[b,2,3],[b,1,8],[b,2,15],[b,3,7],[b,4,29]])
d=np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
def sgn(v):
    if v>=0:
        return 1
    else:
        return -1
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    return oldw+a*(myd-comy(oldw,myx))*myx
i=0
for xn in x:
    w=neww(w,d[i],xn,a)
    i+=1
myx=x[:,1]
myy=x[:,2]
pl.subplot(111)
x_max=np.max(myx)+15
x_min=np.min(myx)-5
y_max=np.max(myy)+50
y_min=np.min(myy)-5
pl.xlabel(u"x")
pl.xlim(x_min, x_max)
pl.ylabel(u"y")
pl.ylim(y_min, y_max)
for i in xrange(0,len(d)):
    if d[i]>0:
        pl.plot(myx[i], myy[i], 'r*')
    else:
        pl.plot(myx[i], myy[i], 'ro')
#绘制测试点
test=np.array([b,9,19])
if comy(w,test)>0:
    pl.plot(test[1],test[2], 'b.')
else:
    pl.plot(test[1],test[2],'bx')
test=np.array([b,9,64])
if comy(w,test)>0:
    pl.plot(test[1],test[2], 'b.')
else:
    pl.plot(test[1],test[2],'bx')
test=np.array([b,9,16])
if comy(w,test)>0:
    pl.plot(test[1],test[2], 'b.')
else:
    pl.plot(test[1],test[2],'bx')
test=np.array([b,9,60])
if comy(w,test)>0:
    pl.plot(test[1],test[2], 'b.')

```

```

else:
    pl.plot(test[1], test[2], 'bx')
#绘制分类线
testx=np.array(range(0,20))
testy=testx*2+1.68
pl.plot(testx,testy,'g--')
pl.show()

```

上面给出的是单样本感知器算法，在这种算法中，每次迭代时，只考虑了用一个训练模式修正权矢量。实际上，可以将几个训练模式一起考虑，而批量修正算法则考虑了使用代价函数来进行分类误差率的控制。批量修正算法要对样本进行多次读取，直到神经网络的误差率降到合适的程度才停止样本的训练，这是它与单样本修正算法本质的区别。算法中的误差率使用最直观的被错分类的样本数量准则。

批量修正算法的核心在于其权值更新策略。批量修正算法采用的是梯度下降原理，其计算公式为：

$$w(k+1)=w(k)-\eta(k) \nabla J(w(k))$$

其中， $\eta(k)$ 称为学习率， $\nabla J(w(k))$ 为梯度，计算方法为：

$$\nabla J(w(k)) = \sum_{y \in Y_k} (-y)$$

其中，

$$Y_k = \text{被错分类的样本输出值集合}$$

最终得出权值更新策略为：

$$w(k+1) = w(k) + \eta(k) \sum_{y \in Y_k} d * y \quad (7-1)$$

在上式中， d 为样本实际输出值， y 为被错分类的样本的输出值。

具体算法过程如下：

- (1) 初始化权值、学习率，以及期望误差率。
- (2) 读取所有样本数据。
- (3) 依次对样本进行训练，更新权值，其更新策略如上式所示。

(4) 检查 $\left| \eta(k) \sum_{y \in Y_k} d * y \right|$ 是否小于指定误差率，或者训练次数是否已到，如果没有达到误差率的要求且训练次数未到，转到第 (2) 步执行。

继续使用单样本修正法的样本，将一组 (x, y) 的输入划分为以下两类函数之一：

- $2x+1=y$ 为第 1 类。
- $7x+1=y$ 为第 2 类。

下面用 Python 实现这个神经网络，最后用 [9,19] 和 [3,22] 对训练成功后的网络进行测试。

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-4.py

```

```

import numpy as np
b=1
a=0.5
x = np.array([[1,1,3],[1,2,3],[1,1,8],[1,2,15]])
d =np.array([1,1,-1,-1])
w=np.array([b,0,0])
wucha=0
ddcount=50
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def tiduxz(myw,myx,mya):
    i=0
    sum_x=np.array([0,0,0])
    for xn in myx:
        if comy(myw,xn)!=d[i]:
            sum_x+=d[i]*xn
        i+=1
    return mya*sum_x
i=0
while True:
    tdxz=tiduxz(w,x,a)
    w=w+tdxz
    i=i+1
    if abs(tdxz.sum())<=wucha or i>=ddcount:break
test=np.array([1,9,19])
print "%d %d => %d "%(test[1],test[2],comy(w,test))
test=np.array([1,3,22])
print "%d %d => %d "%(test[1],test[2],comy(w,test))

```

运行程序后可发现，训练效果很好，测试数据被正确分类。

9 19 => 1

3 22 => -1

3. LMS 算法

LMS 算法全称为 least mean square 算法，中文是最小均方算法。在 ANN 领域内，均方误差是指样本预测输出值与实际输出值之差平方的期望值，记为 MSE 。设 $observed$ 为样本真值， $predicted$ 为样本预测值，计算公式如下：

$$MSE = \frac{1}{n} \sum_{t=1}^n (observed_t - predicted_t)^2$$

LMS 算法的策略是使均方误差最小，该算法运行在一个线性神经元上，使用的是批量修正算法，其误差信号为：

$$e(n)=d(n)-X^T(n)\hat{W}(n)$$

然后在误差信号的基础上计算梯度向量，公式如下：

$$\frac{\partial \theta(n)}{\partial \hat{w}(n)} = -X(n)e(n)$$

最后，生成权值调整方案。公式如下：

$$\hat{W}(n+1) = \hat{W}(n) + \eta X(n)e(n)$$

在上式中， η 表示学习率，该值越小，LMS 算法执行得越精确，但同时算法收敛速度越慢。

下面使用 LMS 算法实现逻辑与运算，将学习率设为 0.1。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-5.py
import numpy as np
b=1
a=0.1
x = np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0]])
d =np.array([1,1,1,0])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=20
def sgn(v):
    if v>0:
        return 1
    else:
        return 0
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye/=float(i)
    mycount+=1
    print u"第 %d 次调整后的权值: "%mycount
    print w
    print u"误差: %f"%mye
    if mye<expect_e or mycount>maxtrycount:break
for xn in x:
```

```
print "%d and %d => %d "%(xn[1], xn[2], get_v(w, xn))
```

下面是程序的运行结果，可以看出，通过 13 次迭代，训练结束，对测试值的输出正确。

```
.....
.....
第 12 次调整后的权值:
[-0.1  0.1  0.1]
误差: 0.500000
第 13 次调整后的权值:
[-0.1  0.1  0.1]
误差: 0.000000
1 and 1 => 1
1 and 0 => 1
0 and 1 => 1
0 and 0 => 0
>>>
```

另外，可应用 LMS 算法实现比逻辑与更复杂的算法，比如：在输入矩阵中，如果 x 向量的整除结果为 6，则表示为一类，输出为 1；若结果为 3 则是另一类，输出为 -1。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-6.py
import numpy as np
b=1
a=0.1
x = np.array([[1,1,6],[1,2,12],[1,3,9],[1,8,24]])
d =np.array([1,1,-1,-1])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=20
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
```

```

        i+=1
        mye+=pow(e, 2)
    mye/=float(i)
    mycount+=1
    print u"第 %d 次调整后的权值: "%mycount
    print w
    print u"误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break
for xn in x:
    print "%d      %d => %d "%(xn[1],xn[2],get_v(w,xn))
test=np.array([1,9,27])
print "%d      %d => %d "%(test[1],test[2],get_v(w,test))
test=np.array([1,11,66])
print "%d      %d => %d "%(test[1],test[2],get_v(w,test))

```

通过 9 次训练后，误差率为 0，用样本数据和测试数据进行验证，准确无误。下面是执行结果：

```

.....
.....
第 8 次调整后的权值:
[ 1.4 -2.8  0.6]
误差: 3.000000
第 9 次调整后的权值:
[ 1.4 -2.8  0.6]
误差: 0.000000
1      6 => 1
2      12 => 1
3      9 => -1
8      24 => -1
9      27 => -1
11     66 => 1

```

4. Rosenblatt 感知器的局限性

不是所有的数据都能被 Rosenblatt 感知器正确分类。比如说下面的样本数据：

```

x = np.array([[1,1,6],[1,3,12],[1,3,9],[1,3,21],[1,2,16],[1,3,15]]) d =np.
array([1,1,-1,-1,1,-1])

```

应用 LMS 算法对这些数据训练 200 次，效果仍非常差，样本数据和测试数据都无法正确分类。如下所示：

```

.....
.....
第 199 次调整后的权值:
[ 18. -17.2   0.8]
误差: 2.666667
第 200 次调整后的权值:
[ 18. -17.4  -0.8]
误差: 2.666667
第 201 次调整后的权值:
[ 18.4 -16.8   1.8]

```

```

误差: 2.666667
1      6 => 1
3      12 => -1
3      9 => -1
3      21 => 1
2      16 => 1
3      15 => -1
9      27 => -1
11     66 => -1

```

为什么会出现这种情况？试着修改 7-6.py 的样本数据，同时绘制包括这些点的散点图。

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-7.py
import numpy as np
import pylab as pl
b=1
a=0.1
x = np.array([[1,1,6],[1,3,12],[1,3,9],[1,3,21],[1,2,16],[1,3,15]])
d =np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=200
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye/=float(i)
    mycount+=1
    print u"第 %d 次调整后的权值: "%mycount
    print w
    print u"误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break

```

```

for xn in x:
    print "%d      %d => %d "%(xn[1],xn[2],get_v(w,xn))
test=np.array([1,9,27])
print "%d      %d => %d "%(test[1],test[2],get_v(w,test))
test=np.array([1,11,66])
print "%d      %d => %d "%(test[1],test[2],get_v(w,test))
myx=x[:,1]
myy=x[:,2]
pl.subplot(111)
x_max=np.max(myx)+10
x_min=np.min(myx)-5
y_max=np.max(myy)+50
y_min=np.min(myy)-5
pl.xlabel(u"x")
pl.xlim(x_min, x_max)
pl.ylabel(u"y")
pl.ylim(y_min, y_max)
#绘制样本点
for i in xrange(0,len(d)):
    if d[i]>0:
        pl.plot(myx[i], myy[i], 'r*')
    else:
        pl.plot(myx[i], myy[i], 'ro')
#绘制测试点
test=np.array([1,9,27])
pl.plot(test[1],test[2], 'bx')
test=np.array([1,11,66])
pl.plot(test[1],test[2], 'bx')
pl.show()

```

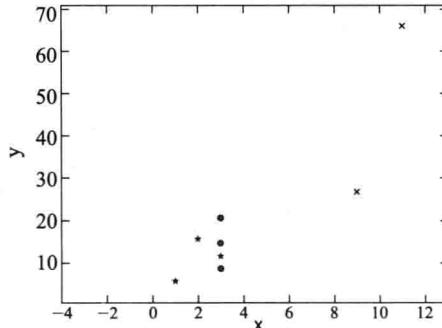


图 7-5 散点图

在如图 7-5 所示的散点图中，实心圆圈和星号是两类不同的样本点，叉号为测试点。在这张图中，无法画出一条线来将两类点分开，只有曲线才能将它们分开，但线性方程根本不可能产生的曲线，因此 Rosenblatt 感知器不适用于非线性分类。

5. LMS 的学习率退火算法

模拟退火算法来源于固体退火原理。退火就是将材料加热后再经特定速率冷却，目的是增大晶粒的体积，并且减少晶格中的缺陷。材料中的原子原来会停留在使内能有局部最小值的位置，加热使能量变大，原子会离开原来位置，而随机在其他位置中移动。退火冷却时速度较慢，徐徐冷却时粒子渐趋有序，原子有可能找到内能比原先更低的位置，最后在常温时达到基态，内能减为最小。

根据 Metropolis 准则，粒子在温度 T 时趋于平衡的概率为 $e^{-\Delta E/(kT)}$ ，其中 E 为温度 T 时的内能， ΔE 为其改变量， k 为 Boltzmann 常数。用固体退火模拟组合优化问题，将内能 E 模拟为目标函数值 f ，温度 T 演化成控制参数 t ，即得到解组合优化问题的模拟退火算法：由初始解 i 和控制参数初值 t 开始，对当前解重复进行“产生新解→计算目标函数差→接受或舍弃”的迭代，并逐步衰减 t 值，算法终止时的当前解即为所得近似最优解，这是基于蒙特卡罗迭代求解法的一种启发式随机搜索过程。退火过程由冷却进度表控制，包括控制参数

的初值 t 及其衰减因子 Δt 、每个 t 值时的迭代次数 L 和停止条件 S 。

针对学习率不变化、收敛速度较慢的情况，即可使用退火算法方案，让学习率随时间而变化。学习率计算公式为：

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)}$$

下面应用退火算法对代码 7-6.py 做一些改动，代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-8.py
import numpy as np
import math
b=1
a0=0.1
a=0.0
r=5.0
x = np.array([[1,1,6],[1,2,12],[1,3,9],[1,8,24]])
d =np.array([1,1,-1,-1])
w=np.array([b,0,0])
expect_e=0.05
maxtrycount=20
mycount=0
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    a=a0/(1+float(mycount)/r)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye=math.sqrt(mye)
    mycount+=1
    print u"第 %d 次调整后的权值: "%mycount
    print w
    print u"误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break
```

```

for xn in x:
    print "%d %d => %d "%(xn[1],xn[2],get_v(w,xn))
test=np.array([1,9,27])
print "%d %d => %d "%(test[1],test[2],get_v(w,test))
test=np.array([1,11,66])
print "%d %d => %d "%(test[1],test[2],get_v(w,test))

```

从程序运行结果来看，仅仅 7 次就完成了训练，训练次数大大减少。

```

.....
.....
第 6 次调整后的权值:
[ 1.28888889 -1.55238095  0.29642857]
误差: 3.464102
第 7 次调整后的权值:
[ 1.28888889 -1.55238095  0.29642857]
误差: 0.000000
1   6 => 1
2   12 => 1
3   9 => -1
8   24 => -1
9   27 => -1
11  66 => 1
>>>

```

7.1.2 梯度下降

1. 梯度下降概述

上一节中曾提到梯度的概念，究竟什么是梯度和梯度下降呢？

梯度是一个向量场，标量场中某一点上的梯度指向标量场增长最快的方向，梯度的长度是这个最大的变化率。梯度下降，就是利用负梯度方向来决定每次迭代的新的搜索方向，从而使得在每次迭代过程中，都能让待优化的目标函数逐步减小。梯度下降法使用的是二范数下的最速下降法。最速下降法的一种简单形式如下：

$$x(k+1) = x(k) - \alpha * g(k)$$

其中， α 称为学习速率，可以是较小的常数。 $g(k)$ 是 $x(k)$ 的梯度。

机器学习算法效果究竟如何，或者说误差为多少，可以用误差函数 $J(\theta)$ 来度量。其中的参数 θ 在神经网络中可以理解为权值。如何调整权值 θ 以使得 $J(\theta)$ 取得最小值有很多方法，梯度下降法是按下面的步骤进行的：

(1) 对 θ 赋值，这个值可以是随机的，也可以让 θ 是一个全零的向量。

(2) 改变 θ 的值，使得 $J(\theta)$ 按梯度下降的方向进行减少。

为了更清楚地表达，先来看一下如图 7-6 所示的误差曲面及梯度下降图。

图 7-6 是表示参数 θ 与误差函数 $J(\theta)$ 关系的图，也称误差曲面图。该图上的方向表明，随着迭代次数的增加，误差走向了曲面的最小误差点。

红色较凸起的部分是表示 $J(\theta)$ 有着比较高的取值，对其进行神经网络训练时，最完美

的目标是：让 $J(\theta)$ 的值尽量低，降到最低处，也就是最凹的部分。 θ_0 、 θ_1 表示 θ 向量的两个维度。

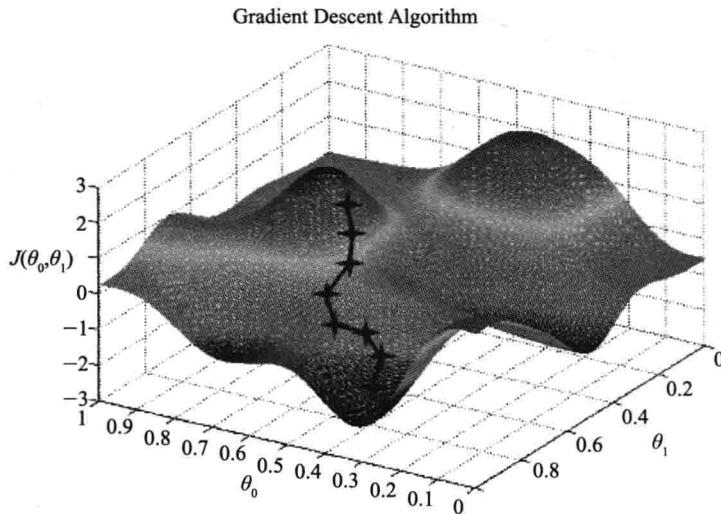


图 7-6 误差曲面及梯度下降 (附彩图)

梯度下降法的第一步是给 θ 一个初值，假设随机给的初值是在最高的十字点处；然后将 θ 按照梯度下降的方向进行调整，就会使得 $J(\theta)$ 往更低的方向变化，如图 7-7 所示。算法的结束将是在 θ 下降到无法继续下降为止。当然，可能梯度下降的最终点并非是全局最小点（图 7-6 的路径中的最后一个点），而是一个局部最小点（图 7-7 的路径中的最后一个点），比如图 7-7 所示的情况，而图 7-6 中的曲线路径则是完美的梯度下降过程。

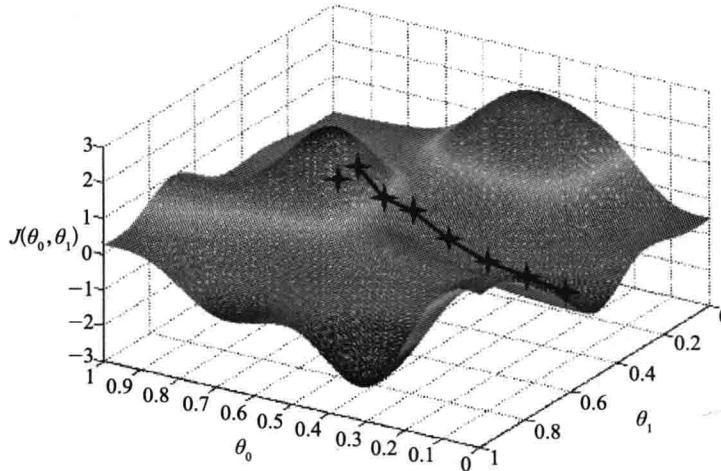


图 7-7 落到局部最小点的梯度下降 (附彩图)

图 7-7 中所示的这种情况是在神经网络训练中要尽量避免出现的，这里的梯度下降到局

部最小点后停止，神经网络在一个不正确的位置收敛了，训练停止，这时即使继续训练也没有任何效果。

2. 梯度下降法涉及的数学知识

(1) 导数的几何意义：导数的几何意义在于，函数在某一点的导数等于它的图像上这一点处之切线的斜率，如图 7-8 所示。

其斜率为：

$$\tan \alpha = \lim_{\Delta x \rightarrow 0} \tan \varphi = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

(2) 积分的几何意义：对于一个给定的正实值函数 $f(x)$ ， $f(x)$ 在一个实数区间 $[a, b]$ 上的定积分为：

$$\int_a^b f(x) dx$$

定积分的几何意义可以理解为在 Oxy 坐标平面上，由曲线 $(x, f(x))$ 、直线 $x=a, x=b$ 以及 x 轴围成的曲边梯形的面积，这个面积可以是一块面积，也可以是多块面积的和（注意，这里的面积有正负之分）。

(3) 微分的几何意义：一元函数的微分与自变量的微分之商等于该函数的导数，所以导数也叫做微商。

设 Δx 是曲线 $y=f(x)$ 上的点 P 在横坐标上的增量， Δy 是曲线在点 P 处相对于 Δx 的纵坐标增量， dy 是曲线在点 P 的切线对应 Δx 在纵坐标上的增量，如图 7-9 所示。

(4) 偏导数：拥有多个自变量的函数的偏导数是指在保持其他自变量恒定的情况下，该函数相对于其中某个自变量的导数。函数 f 关于变量 x 的偏导数写为 f'_x 或 $\frac{\partial f}{\partial x}$ 。

f 是一个多元函数。例如：

$$f(x, y) = x^2 + xy + y^2$$

因为曲面上的每一点都有无穷多条切线，描述这种函数的导数相当困难。偏导数就是选择其中一条切线，并求出它的斜率。

多自变量函数 $f(x_1, \dots, x_n)$ 在点 (a_1, \dots, a_n) 处，对于某个自变量 x_i 的偏导数，其定义为：

$$\frac{\partial f}{\partial x_i}(a_1, \dots, a_n) = \lim_{h \rightarrow 0} \frac{f(a_1, \dots, a_i + h, \dots, a_n) - f(a_1, \dots, a_n)}{h}$$

(5) 梯度：欧几里得空间 \mathbb{R}^n 中的函数 $f(x_1, \dots, x_n)$ 对每个自变量 x_j 具有偏导数 $\partial f / \partial x_j$ ，在点 a 处，这些偏导数定义了一个向量：

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \dots, \frac{\partial f}{\partial x_n}(a) \right)$$

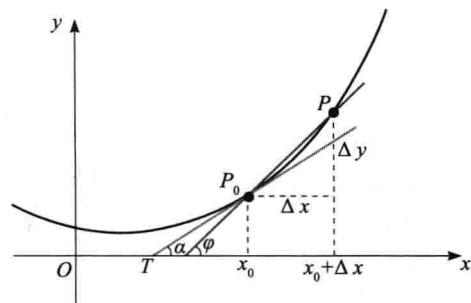


图 7-8 导数

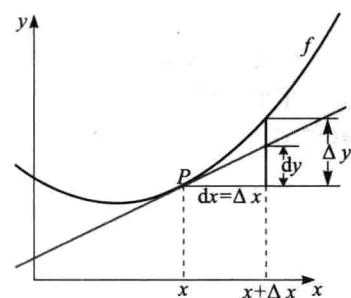


图 7-9 微分

这个向量称为 f 在点 a 处的梯度。

3. 梯度下降的原理

如果实值函数 $F(x)$ 在点 a 处可微且有定义，那么该函数在 a 点沿着梯度相反的方向下降最快，从函数 F 对局部极小值的初始估计值 x_0 出发，存在如下序列 x_0, x_1, x_2, \dots ，使得

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n), n \geq 0$$

因此可得到

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots$$

如果顺利的话，序列 (X_n) 将收敛到期望的极值。

4. 梯度下降和 delta 法则

delta 法则克服了感应器法则的不足，在线性不可分的训练样本上，可以有效地收敛，并接近目标的最佳近似值。delta 法则的关键思想是：使用梯度下降来搜索可能的权向量假设空间，以找到最佳拟合训练样例的权向量。

delta 法则为反向传播算法提供了基础，而反向传播算法能够学习多个单元的互连网络，在包含多种不同类型的连续参数化假设的空间中，梯度下降是必须遍历这样的空间的所有算法的基础。

误差曲面是一个抛物面，存在一个单一全局最小值，梯度下降搜索从一个任意的初始权向量开始，然后沿误差曲面最陡峭下降的方向，以很小的步伐反复修改这个向量，直到得到全局的最小误差点。

5. 基于梯度下降的线性分类器

下面总结一下前面介绍的线性神经网络代码，并将它们整理成 Python 模块 `mplannliner`，然后引入这个模块，实现对一组输入的线性分类。以下程序有详细的注释说明。

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-9.py
import mplannliner as nplann
traindata1=[[[9,25],-1],[[5,8],-1],[[15,31],-1],[[35,62],-1],[[19,40],-1],[[28,65],1],[[20,59],1],[[9,41],1],[[12,60],1],[[2,37],1]]
myann=nplann.Mplannliner()
#样本初始化
myann.samples_init(traindata1)
#学习率初始化
myann.a_init(0.1)
#搜索时间常数初始化
myann.r_init(50)
#最大训练次数
myann.maxtry_init(500)
#期望最小误差
myann.e_init(0.05)
#训练
```

```

myann.train()
#仿真，测试，对未知样本分类
myc=myann.simulate([35,68])
print "[35,68]"
if myc==1:
    print u"正类"
else:
    print u"负类"
#将测试点在最终效果图上显示出来，将它加入drawponint集，测试点表现为"**"，并且色彩由其最终的分类结果而决定
myann.drawponint_add([35,68])
myc=myann.simulate([35,82])
print "[35,82]"
if myc==1:
    print u"正类"
else:
    print u"负类"
myann.drawponint_add([35,82])
myann.draw2d()

#下面直接使用默认参数进行训练
traindata2=[[9,25,30],-1],[[5,8,12],-1],[[15,31,49],-1],[[35,62,108],
-1],[[19,40,60],-1],[[28,65,98],1],[[20,59,72],1],[[9,41,38],1],[[12,60,46],1],[[2,37,
18],1]]
myann2=nplann.Mplannliner()
myann2.samples_init(traindata2)
myann2.train()
myc=myann2.simulate([35,68,110])
print "[35,68,110]"
if myc==1:
    print u"正类"
else:
    print u"负类"

```

如图 7-10 所示是程序运行生成的效果图，虚线是分类线，它清晰地将不同类的点分成了上下两部分。星号为测试数据，可以看到，已被正确分类。Rosenblatt 感知器对线性分类效果还是不错的。

`mplannliner` 模块的代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#author:麦好
#2013-07-25
import numpy as np
import math
import matplotlib.pyplot as plt
class Mplannliner:

```

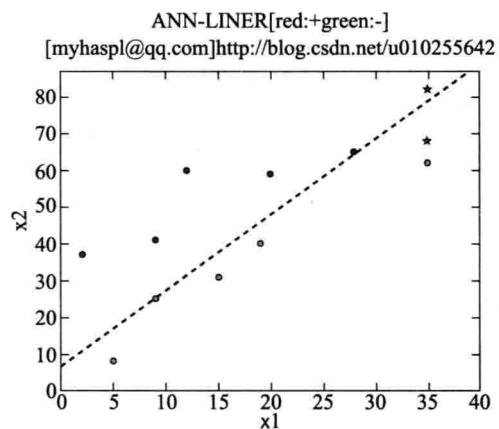


图 7-10 线性分类效果

```

def __init__(self):
    self.b=1
    self.a0=0.1
    self.a=0.0
    self.r=20.0
    self.expect_e=0.05
    self.traincount=100
    self.testpoint=[]
def testpoint_init(self):
    self.testpoint=[]
def e_init(self,mye):
    self.expect_e=mye
def samples_init(self,mysamples):
    my_x = []
    my_d = []
    my_w =[self.b]
    myexamp=mysamples
    for mysmp in myexamp:
        tempsmp=[1]+mysmp[0]
        my_x.append(tempsmp)
        my_d.append(mysmp[1])
    for i in range(len(my_x[0])-1):
        my_w.append(0.0)
    self.x = np.array(my_x)
    self.d = np.array(my_d)
    self.w = np.array(my_w)
def a_init(self,mya):
    self.a0=mya
def r_init(self,myr):
    self.r=myr
def maxtry_init(self,maxc):
    self.traincount=maxc
def sgn(self,v):
    if v>0:
        return 1
    else:
        return -1
def get_v(self,myw,myx):
    return self.sgn(np.dot(myw.T,myx))
def neww(self,oldw,myd,myx,a,mycount):
    mye=self.get_e(oldw,myx,myd)
    self.a=self.a0/(1+mycount/float(self.r))
    return (oldw+a*mye*myx,mye)
def get_e(self,myw,myx,myd):
    return myd-self.get_v(myw,myx)
def train(self):
    mycount=0
    while True:
        mye=0
        i=0

```

```

        for xn in self.x:
            self.w,e=self.neww(self.w,self.d[i],\
            xn,self.a,mycount)
            i+=1
            mye+=pow(e,2)
            mye=math.sqrt(mye)
            mycount+=1
            print u"第%d次调整中...误差: %f"%(mycount,mye)
            if abs(mye)<self.expect_e or mycount>self.traincount:
                if mycount>self.traincount:
                    print "已经达到最大训练次数:%d"%mycount
                    break
    def simulate(self,testdata):
        if self.get_v(self.w,np.array([1]+testdata))>0:
            return 1
        else:
            return -1
    def drawponint_add(self,point):
        self.testpoint.append(point)

    def draw2d(self):
        temp_x=[]
        temp_y=[]
        i=0
        for mysamp in self.x:
            temp_x.append(mysamp[1])
            temp_y.append(mysamp[2])
            if self.d[i] > 0:
                plt.plot(mysamp[1],mysamp[2],"or")
            else:
                plt.plot(mysamp[1],mysamp[2],"og")
            i+=1
        mytestpointx=[]
        mytestpointy=[]
        for addpoint in self.testpoint:
            if self.simulate(addpoint)=="+":
                plt.plot(addpoint[0],addpoint[1], '*r')
            else:
                plt.plot(addpoint[0],addpoint[1], '*g')

        mytestpointx.append(addpoint[0])
        mytestpointy.append(addpoint[1])

        x_max=max(max(temp_x),max(mytestpointx))+5
        x_min=min(min(temp_x),min(mytestpointx))
        y_max=max(max(temp_y),max(mytestpointy))+5
        y_min=min(min(temp_y),min(mytestpointy))
        if x_min >0:
            x_min=0
        if y_min >0:

```

```

y_min=0
plt.xlabel(u"x1")
plt.xlim(x_min, x_max)
plt.ylabel(u"x2")
plt.ylim(y_min, y_max)
plt.title("ANN-LINER[red:+ green:-] " )
lp_x1 = [x_min, x_max]
lp_x2 = []
myb=self.w[0]
myw1=self.w[1]
myw2=self.w[2]
myy=(-myb-myw1*lp_x1[0])/float(myw2)
lp_x2.append(myy)
myy=(-myb-myw1*lp_x1[1])/float(myw2)
lp_x2.append(myy)
plt.plot(lp_x1, lp_x2, 'b--')
plt.show()

```

7.1.3 反向传播与多层感知器

1. 反向传播算法

反向传播算法对梯度下降法进行了改进，主要由两个环节（激励传播、权重更新）反复循环迭代，直到网络对输入的响应达到预定的目标范围为止。它可用来学习多层网络的权值，通过搜索一个巨大的假设空间（这个空间由网络中所有单元的所有可能权值定义）得到误差曲面。在多层神经网络中，误差曲面存在全局最小值，但也可能存在多个局部极小值，梯度下降法不能保证收敛到全局极小值，但反向传播算法较好地解决了这个问题，在实践中有出色的效果。反向传播算法有以下特点：

(1) 网络中的每个神经元模型包括一个非线性激活函数，非线性是光滑的（即处处可微）也是必要的，否则网络的输入输出关系会被归结为单层感知器。

(2) 网络包括一层或多层神经元的隐层，它不负责网络的输入输出。这些隐层神经元逐步从输入向量中提取有用特征，使网络学习复杂的任务。

(3) 网络的连接强度由网络突触决定。

反向传播算法的主要过程如下：

(1) 激励传播过程。在神经元 j 的激活函数输入处应用诱导局部域 $v_j(n)$ ，定义如下：

$$v_j(n) = \sum_{i=0}^m w_{ji}(n)y_i(n)$$

这个过程完成逐层从输入层传播至输出层的任务。

(2) 权重更新过程。对每层的神经元的权值进行修正，公式如下：

$$\Delta w_{ji}(n) = \eta \delta_j(n)y_i(n)$$

其中， $\Delta w_{ji}(n)$ 是指神经元 i 连接到神经元 j 的突触 ji 的权值的校正值， η 是学习率， $\delta_j(n)$ 是局部梯度。

反向传播算法的过程如下：

从建立一个具有期望数量的隐藏单元和输出单元的网络开始，初始化所有的网络的权值为较小的随机数（0~1的实数）；然后，给定一个固定的网络结构；最后，算法的主循环对训练样例进行反复迭代，对于每一个训练样例，它会应用目前的网络到这个样例中，并计算出对这个样例网络输出的误差，更新网络中所有的权值，然后对这样的梯度下降步骤进行迭代，直到网络的性能达到可接受的精度为止。

但该算法存在局限性，仍然无法摆脱陷入局部最小误差的地步。因此，要在此基础上增加冲量（动量）项。通过修改权值更新法则，使第n次迭代时权值的更新受到第n-1次迭代的影响，即本次迭代的更新值的计算，有部分参数来自于上次迭代的更新值，这样，冲量有时会滚过误差曲面的局部极小值，并在梯度不变的区域内逐渐增大搜索步长，加快收敛，从而减少训练次数。

2. 多层感知器网络

多层感知器利用非线性神经元作为中间隐藏层神经元，输出端可以直接使用非线性神经元，也可以使用线性神经元。如图7-11所示是一个拥有两个隐藏层的多层感知器网络。

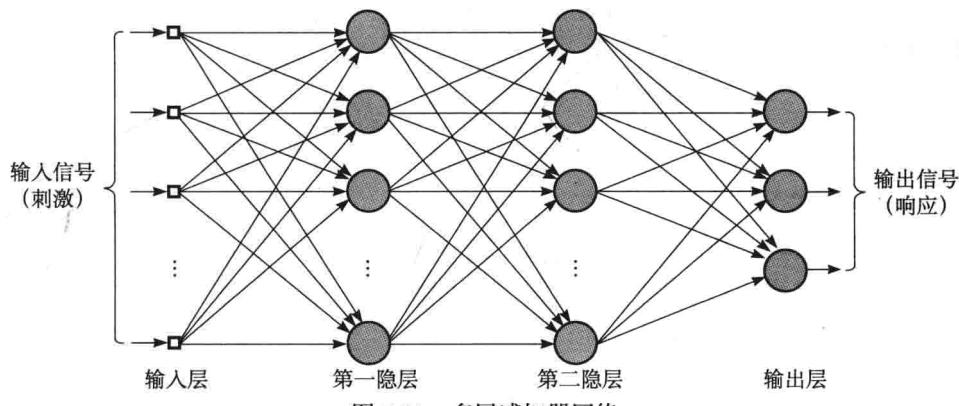


图 7-11 多层感知器网络

在图7-11中，网络由输入层、隐藏层及输出层构成，它能够表示种类繁多的非线性曲面，多层次能在隐藏层自动发现并被有效表示。它允许学习器创造出设计者没有明确引入的特征，网络中使用的单元层越多，可以创造出的特征越复杂。

多层次感知器的激活函数及局部梯度如下。

(1) logistic 函数。logistic 函数的定义为：

$$P(t) = \frac{1}{1+e^{-t}}$$

它又称 sigmoid 曲线 (S型曲线)，如图7-12所示是它的图像，很像字母 S。

在多层次感知器中，logistic 函数可作为神经元 j 的激活函数 $\varphi'_j(v_j(n))$ ，它定义为：

$$\varphi'_j(v_j(n)) = \frac{1}{1+\exp(-av_j(n))}, a > 0$$

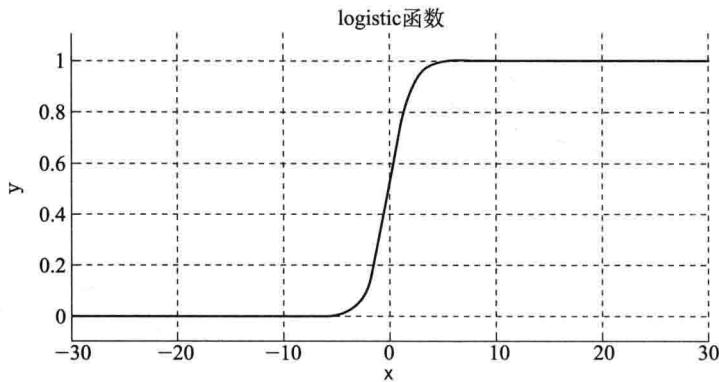


图 7-12 sigmoid 曲线 (附彩图)

其中， $v_j(n)$ 是神经元 j 的诱导局部域。

logistic 函数的局部梯度需要分两种情况定义：

第一种情况，神经 j 没有位于隐藏层，则局部梯度定义为：

$$\delta_j(n) = a(d_j(n) - o_j(n))o_j(n)(1 - o_j(n))$$

第二种情况，神经 j 位于隐藏层，则局部梯度定义为：

$$\delta_j(n) = ay_j(n)(1 - y_j(n)) \sum_k \delta_k(n)w_{kj}(n)$$

(2) tanh 函数。tanh (双曲正切) 函数的数学定义为：

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\tanh x = \frac{\sinh(x)}{\cosh(x)}$$

双曲正切函数的图像如图 7-13 所示。

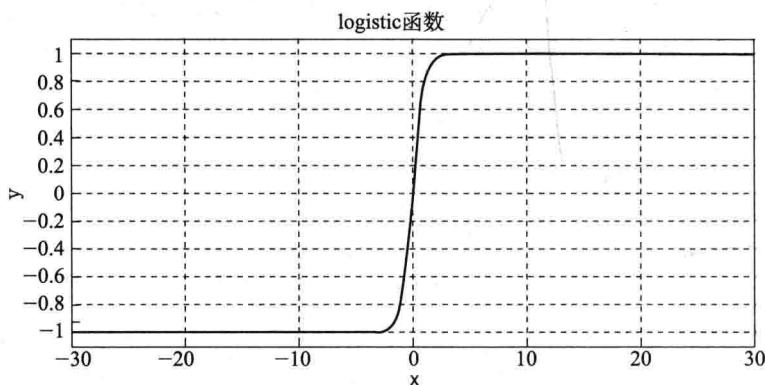


图 7-13 双曲正切函数

在多层感知器中，双曲正切函数可作为神经元 j 的激活函数 $\varphi_j(v_j(n))$ ，它定义为：

$$\varphi_j(v_j(n)) = \operatorname{atanh}(bv_j(n))$$

其中， a 和 b 为正的常数值， $v_j(n)$ 是神经元 j 的诱导局部域。

双曲正切函数的局部梯度需要分两种情况定义：

第一种情况，神经元 j 没有位于隐藏层，则局部梯度定义为：

$$\delta_j(n) = \frac{b}{a} (d_j(n) - o_j(n))(a - o_j(n))a(a + o_j(n))$$

第二种情况，神经 j 位于隐藏层，则局部梯度定义为：

$$\delta_j(n) = \frac{b}{a} (d_j(n) - y_j(n))(a + y_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

前面说过，多层感知器利用非线性神经元作为中间隐藏层神经元，非线性神经元使用双曲正切函数或 logistic 函数作为激活函数，使用非线性神经元或使用线性神经元组成输出层。多层感知器的训练需要很多样本，其中每个样本的学习过程分为前向计算和反向计算。

前向计算根据权值矩阵由前向后一层层神经元地推进，每次推进根据权值计算每层的输出值；反向计算根据输出结果与目标输出的误差来由后向前调整神经网络的权值，引入冲量作为神经网络权值调整的依据之一，冲量（动量）参数既可以调整学习速度，还能体现时间延迟。

整个算法过程也是反向传播算法的过程，通过使用梯度下降方法搜索可能假设的空间，迭代减小网络的误差以拟合训练数据。

如果不调用神经网络的中间库，自己编写所有的代码，需要把握好以下要点：

(1) 学习率和动量参数。输入层、输出层、中间层的学习率和动量参数不同。输出层的学习率较低，动量参数较高；输入层的学习率较低，动量参数较低。

(2) 权值。权值矩阵非常重要，一个好的权值矩阵能使网络快速收敛，让网络更稳定。关于权值初始化的策略，可以选择以下几种方法：

□ 随机初始化。

□ 逐步搜索法。

□ Nguyen-Widrow 初始化算法，MATLAB 使用 Nguyen-Widrow 权值矩阵初始化算法。

(3) 输出层处理，为了确保输出层输出的数据符合预测结果的要求，需要另外的函数对输出层进行处理。在线性神经网络中，通常使用硬限幅函数，在多层感知器这种非线性神经网络中既可以使用硬限幅，也可以使用其他函数，具体使用什么函数要看训练效果如何。

(4) 数据预处理。输入数据五花八门，在训练之前对数据进行预处理是非常必要的。通过预处理权值矩阵使进入神经网络的数值不会过大或过小，以保证通过中间层的非线性神经元时，输出不逼近其极限。

先来看图 7-14，上面显示的数据散乱且不均匀。

数据预处理的目标是：将图 7-14 中的数据转化为如图 7-15 所示的形式，数据均匀地分布在坐标系中，4 个象限均有数据存在。

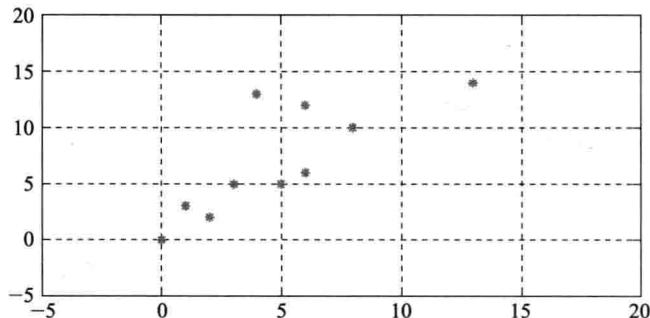


图 7-14 散乱的数据

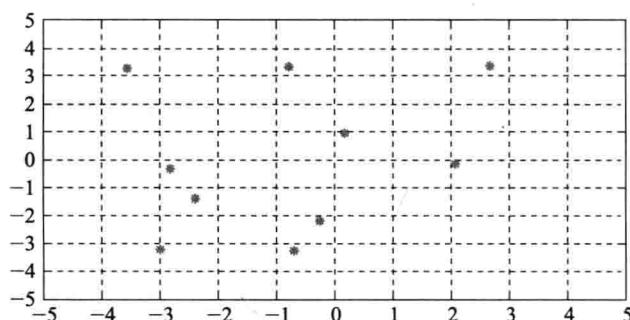


图 7-15 均匀分布的数据

前面讲述了多层次感知器的理论基础，下面用 Python 实现一个多层次感知器，理论联系实践，这样才能更好地理解反向传播算法和多层次感知器。这里要实现的是使用感知器将一组数据进行非线性分类，具体要求为：对下面这组输入数据和输出目标进行训练，对未知数据进行仿真测试。

```
x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
```

为了简化实现过程，使用原始的纯随机生成方法产生多层次感知器网络的权值矩阵。权值初始值既要保证权值矩阵实现输入项在网络中均匀分布，又要保证权值矩阵本身的均匀分布。为达到这个目的，随机生成若干个权值矩阵，然后从中选择最优化的权值矩阵。最优化的标准为：

(1) 对于输入层的权值设计，要尽量使输入数据的方差接近 1。此外，对于相差较大的样本，将它们处理为分布不太接近饱和的可供训练的输入数据。

(2) 权值矩阵的均值要尽可能小，其方差尽可能与神经元的突触连接数成反比。

同时，要考虑到数据的预处理，需要在输入层前设置一个预处理权值矩阵，所有的输入经过预处理权值矩阵处理后进入多层次感知器的输入层。代码如下：

```
#对输入数据进行预处理
ann_max=[]
```

```

for m_ani in xrange(0,warray_tnx):
    temp_x=np.array(train_x)
    ann_max.append(np.max(temp_x[:,m_ani]))
ann_max=np.array(ann_max)
def getnowsx(mysx,in_w):
    '''生成本次的扩维输入数据'''
    global warray_n
    mysx=np.array(mysx)
    x_end=[]
    for i in xrange(0,warray_n):
        x_end.append(np.dot(mysx,in_w[:,i]))
    return x_end

def get_inlw(my_train_max,w_count,myin_x):
    '''计算对输入数据预处理的权值'''
    #对随机生成的多个权值进行优化选择，选择最优的权值
    global warray_tnx
    global warray_n
    mylw=[]
    y_in=[]
    #生成测试权值
    mylw=np.random.rand(w_count,warray_tnx,warray_n)
    for ii in xrange(0,warray_tnx):
        mylw[:,ii,:]=mylw[:,ii,:]*1/float(my_train_max[ii])-\
        1/float(my_train_max[ii])*0.5
    #计算输出
    for i in xrange(0,w_count):
        y_in.append([])
        for xj in xrange(0,len(myin_x)):
            y_in[i].append(getnowsx(myin_x[xj],mylw[i]))
    #计算均方差
    mymin=10**5
    mychoice=0
    for i in xrange(0,w_count):
        myvar=np.var(y_in[i])
        if abs(myvar-1)<mymin:
            mymin=abs(myvar-1)
            mychoice=i
    #返回数据整理的权值矩阵
    return mylw[mychoice]

mylnww=get_inlw(ann_max,300,train_x)

def get_inputx(mytrain_x,myin_w):
    '''将训练数据通过输入权数，扩维后形成输入数据'''
    end_trainx=[]
    for i in xrange(0,len(mytrain_x)):
        end_trainx.append(getnowsx(mytrain_x[i],myin_w))
    return end_trainx
x=get_inputx(train_x,mylnww)
def get_siminx(sim_x):

```

```

global mylnww
myxx=np.array(sim_x)
return get_inputx(myxx,mylnww)
def getlevelw(myin_x,wo_n,wi_n,w_count):
    '''计算一层的初始化权值矩阵'''
    mylw=[]
    y_in=[]
    #生成测试权值
    mylw=np.random.rand(w_count,wi_n,wo_n)
    mylw=mylw*2.-1
    #计算输出
    for i in xrange(0,w_count):
        y_in.append([])
        for xj in xrange(0,len(myin_x)):
            x_end=[]
            for myii in xrange(0,wo_n):
                x_end.append(np.dot(myin_x[xj],mylw[i,:,:myii]))
            y_in[i].append(x_end)
    #计算均方差
    mymin=10**3
    mychoice=0
    for i in xrange(0,w_count):
        myvar=np.var(y_in[i])
        if abs(myvar-1)<mymin:
            mymin=abs(myvar-1)
            mychoice=i
    #返回数据整理的权值矩阵
    csmylw=mylw[mychoice]
    return csmylw,y_in[mychoice]

ann_w=[]
def init_annw():
    global x
    global hidelevel_count
    global warray_n
    global d
    global ann_w
    ann_w=[]
    lwyiii=np.array(x)
    for myn in xrange(0,hidelevel_count):
        #层数
        ann_w.append([])
        if myn==hidelevel_count-1:
            for iii in xrange(0,warray_n):
                ann_w[myn].append([])
                for jjj in xrange(0,warray_n):
                    ann_w[myn][iii].append(0.0)
        elif myn==hidelevel_count-2:
            templw,lwyiii=getlevelw(lwyiii,len(d[0]),warray_n,200)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])

```

```

        for xjj in xrange(0,len(d[0])):
            ann_w[myn][xii].append(templw[xii,xjj])
        for xjj in xrange(len(d[0]),warray_n):
            ann_w[myn][xii].append(0.0)
    else:
        templw,lwyii=getlevelw(lwyii,warray_n,warray_n,200)
        for xii in xrange(0,warray_n):
            ann_w[myn].append([])
            for xjj in xrange(0,warray_n):
                ann_w[myn][xii].append(templw[xii,xjj])
    ann_w=np.array(ann_w)
def generate_lw(trycount):
    global ann_w
    print u"产生权值初始矩阵",
    meanmin=1
    myann_w=ann_w
    alltry=30
    tryc=0
    while tryc<alltry:
        for i_i in range(trycount):
            print ".",
            init_annw()
            if abs(np.mean(np.array(ann_w)))<meanmin:
                meanmin=abs(np.mean(np.array(ann_w)))
                myann_w=ann_w
            tryc+=1
            if abs(np.mean(np.array(myann_w)))<0.008:break
    ann_w=myann_w
    print
    print u"权值矩阵平均:%f"%(np.mean(np.array(ann_w)))
    print u"权值矩阵方差:%f"%(np.var(np.array(ann_w)))
generate_lw(15)

```

此外，只需要输出一个值，在这里不使用硬限幅函数，而是返回最大值的索引，因此需要编写对输出值进行最后加工的函数（注意，为了由浅入深讲解，从现在直到后面明确声明，误差率的计算以最后此函数的输出结果为标准）。该函数的定义如下：

```

def o_func(myy):
    myresult=[]
    for i in xrange(0,len(myy)):
        mean=np.mean(myy)
        if myy[i]>mean:
            myresult.append(1.0)
        else:
            myresult.append(0.0)
    return np.array(myresult)

```

另外，可选择 `tanh` 函数作为非线性神经元的激活函数，它的输出值在 [-1,1] 范围内。下面代码完成激活函数（`ann_atanh` 函数）及其局部梯度（`ann_delta_atanh` 函数）的计算。

```
def ann_atanh(myv):
```

```

atanh_a=1.7159#>0
atanh_b=2/float(3)#>0
temp_rs=atanh_a*np.tanh(atanh_b*myv)
return temp_rs

def ann_delta_atanh(myy,myd,nowlevel,level,n,mydelta,myw):
    anndelta=[]
    anndelta=[]
    anndelta_a=1.7159#>0
    anndelta_b=2/float(3)#>0
    if nowlevel==level:
        #输出层
        anndelta=(float(atanh_b)/atanh_a)*(myd-myy)*(atanh_a-myy)*(atanh_a+myy)
    else:
        #隐藏层
        anndelta=(float(atanh_b)/atanh_a)*(atanh_a-myy)*(atanh_a+myy)
        temp_rs=[]
        for j in xrange(0,n):
            temp_rs.append(sum(myw[j]*mydelta))
        anndelta=anndelta*temp_rs
    return anndelta

```

下面介绍反向传播的核心算法，算法分为前向计算和反向计算两个过程。代码如下：

```

def sample_train(myx,myd,n,sigmoid_func,delta_sigfun):
    '''一个样本的前向和后向计算'''
    global ann_yi
    global ann_delta
    global ann_w
    global ann_wj0
    global ann_y0
    global hidelevel_count
    global alllevel_count
    global learn_r
    global train_a
    global ann_oldw
    level=hidelevel_count
    alllevel=alllevel_count
    #清空yi输出信号数组
    hidelevel=hidelevel_count
    alllevel=alllevel_count
    for i in xrange(0,alllevel):
        #第一维是层数，从0开始
        for j in xrange(0,n):
            #第二维是神经元
            ann_yi[i][j]=0.0
    ann_yi=np.array(ann_yi)
    yi=ann_yi
    #清空delta矩阵
    for i in xrange(0,hidelevel-1):
        for j in xrange(0,n):
            ann_delta[i][j]=0.0
    delta=ann_delta

```

```

#保留W的拷贝，以便下一次迭代
ann_oldw=copy.deepcopy(ann_w)
oldw=ann_oldw
#前向计算
if isdebug:print u"前向计算中..."
#对输入变量进行预处理
myo=np.array([])
for nowlevel in xrange(0,alllevel):
    #一层层向前计算
    #计算诱导局部域
    my_y=[]
    myy=yi[nowlevel-1]
    myw=ann_w[nowlevel-1]
    if nowlevel==0:
        #第一层隐藏层
        my_y=myx
        yi[nowlevel]=my_y
    elif nowlevel==(alllevel-1):
        #输出层
        my_y=o_func(yi[nowlevel-1,:len(myd)])
        yi[nowlevel,:len(myd)]=my_y
    elif nowlevel==(hidelvel-1):
        #最后一层输出层
        for i in xrange(0,len(myd)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel,:len(myd)]=my_y
    else:
        #中间隐藏层
        for i in xrange(0,len(myy)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel]=my_y
if isdebug:
    print u"*****本样本训练的输出矩阵*****"
    print yi
    print u"*****"
#计算误差与均方误差
#因为线性输出层为直接复制，所以取非线性隐藏输出层的结果
myo=yi[hidelvel-1][:len(myd)]
myo_end=yi[alllevel-1][:len(myd)]
mymse=get_e(myd,myo_end)
#反向计算
#输入层不需要计算delta,输出层不需要计算W
if isdebug:print u"反向计算中..."
#计算delta
for nowlevel in xrange(level-1,0,-1):
    if nowlevel==level-1:
        mydelta=delta[nowlevel]
        my_n=len(myd)

```

```

        else:
            mydelta=delta[nowlevel+1]
            my_n=n
            myw=ann_w[nowlevel]
            if nowlevel==level-1:
                #输出层
                mydelta=delta_sigfun(myo,myd,None,None,None,None,None)
            ## 
            elif nowlevel==level-2:
                #输出隐藏层的前一层, 传输相当于输出隐藏层的神经元数目的数据
                mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
1,my_n,mydelta[:len(myd)],myw[:,len(myd)])
            else:
                mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
1,my_n,mydelta,myw)

            delta[nowlevel][:my_n]=mydelta
            #计算与更新权值W
            for nowlevel in xrange(level-1,0,-1):
                #每个层的权值不一样
                if nowlevel==level-1:
                    #输出层
                    my_n=len(myd)
                    mylearn_r=learn_r*0.8
                    mytrain_a=train_a*1.6
                elif nowlevel==1:
                    #输入层
                    my_n=len(myd)
                    mylearn_r=learn_r*0.9
                    mytrain_a=train_a*0.8
                else:
                    #其他层
                    my_n=n
                    mylearn_r=learn_r
                    mytrain_a=train_a

            pre_level_myy=yi[nowlevel-1]
            pretrain_myww=oldw[nowlevel-1]
            pretrain_myw=pretrain_myww[:,my_n]

            #第二个调整参数
            temp_i=[]

            for i in xrange(0,n):
                temp_i.append([])
                for jj in xrange(0,my_n):
                    temp_i[i].append(mylearn_r*delta[nowlevel,jj]*pre-
level_myy[i])
            temp_rs2=np.array(temp_i)
            temp_rs1=mytrain_a*pretrain_myw

```

```

#总调整参数
temp_change=temp_rsl+temp_rs2
my_ww=ann_w[nowlevel-1]
my_ww[:, :my_n] += temp_change
if isdebug:
    print "===="
    print u"***权值矩阵***"
    print ann_w
    print u"***梯度矩阵***"
    print delta
    print "===="
return mymse

```

还需要训练神经网络，并读取测试数据，验证效果。其中，训练神经网络的代码如下：

```

train()
delta_sigfun=ann_delta_atanh
sigmoid_func=ann_atanh
i=0
for xn in xrange(0,len(x)):
    print u"样本: %d~%d => "%(train_x[xn][0],train_x[xn][1])
    print simulate(x[xn],sigmoid_func,delta_sigfun)
    print u"=====正确目标值====="
    print d[i]
    i+=1

```

验证神经网络的代码如下：

```

test=np.array(get_siminx([[8,70]]))
print u"测试值: %f      %f "%(8,70)
print simulate(test,sigmoid_func,delta_sigfun)
print u"正确目标值:[1,0]"
test=np.array(get_siminx([[6.5,272]]))
print u"测试值: %f      %f "%(6.5,272)
print simulate(test,sigmoid_func,delta_sigfun)
print u"正确目标值:[0,1]"
exitstr=raw_input(u"按回车键退出")

```

运行上述程序，经过 78 次训练，神经网络达到了训练目标，误差率为 0。从以下运行结果来看，效果不错。

```

.....
.....
-----开始第76次训练----- # # # # # 误差为: 0.816497
-----开始第77次训练----- # # # # # 误差为: 0.577350
-----开始第78次训练----- # # # # # 误差为: 0.000000
训练成功，正在进行检验
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中

```

```

训练成功,误差为: 0.000000
样本: 4~11 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 7~340 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 10~95 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 3~29 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 7~43 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 5~128 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
测试值: 8.000000      70.000000
仿真计算中
[ 1.  0.]
正确目标值:[1,0]
测试值: 6.500000      272.000000
仿真计算中
[ 0.  1.]
正确目标值:[0,1]
按回车键退出

```

上述多层感知器的完整源代码见本书资源包中的“多层感知器神经网络源代码.doc”文件或 7-10.py 文件。

前面用例子说明了 Rosenblatt 感知器的局限性，现在在多层感知器上对 Rosenblatt 感知器无法分类的数据进行测试。将“多层感知器神经网络源代码.doc”文件中的代码中的样本数据进行修改，并加上绘制散点图的代码。

```

#x和d样本初始化
train_x = np.array([[1,6],[3,12],[3,9],[3,21],[2,16],[3,15]])
d = np.array([[1,0],[1,0],[0,1],[0,1],[1,0],[0,1]])

```

```

.....
.....
for xn in xrange(0, len(x)):
    if simulate(x[xn], sigmoid_func, delta_sigfun)[0] > 0:
        pl.plot(train_x[xn][0], train_x[xn][1], "bo")
    else:
        pl.plot(train_x[xn][0], train_x[xn][1], "b*")
pl.show()

```

运行代码对神经网络进行训练，训练成功后，误差为 0。如图 7-16 所示为训练效果，实心圆圈与星号分别表示两类样本点，多层感知器网络的非线性分类成功完成了 Rosenblatt 感知器无法完成的任务。

现在在上述代码基础上，加上若干随机数据点，将学习率设得较小，进一步观察多层感知器的非线性分类能力。代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-11.py
import pylab as pl
import numpy as np
import random
import copy
isdebug=False
#x和d样本初始化
train_x = np.array([[1,6],[3,12],[3,9],[3,21],[2,16],[3,15]])
d = np.array([[1,0],[1,0],[0,1],[0,1],[1,0],[0,1]])
.....
.....
train()
delta_sigfun=ann_delta_atanh
sigmoid_func=ann_atanh
temp_x=np.random.rand(20)*10
temp_y=np.random.rand(20)*20+temp_x
myx=temp_x
myy=temp_y
pl.subplot(111)
x_max=np.max(myx)+5
x_min=np.min(myx)-5
y_max=np.max(myy)+5
y_min=np.min(myy)-5
pl.xlabel(u"x1")
pl.xlim(x_min, x_max)
pl.ylabel(u"x2")
pl.ylim(y_min, y_max)
i=0
for mysamp in myx:

```

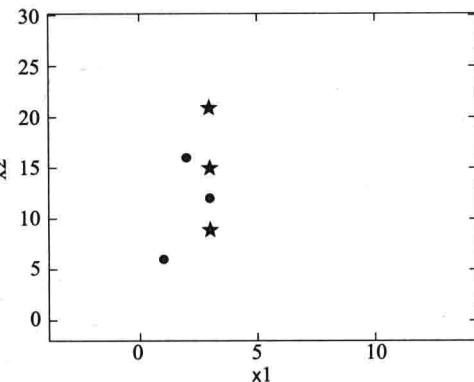


图 7-16 多层感知器网络的非线性分类

```

test=get_siminx([[mysamp,myy[i]]])
if simulate(test,sigmoid_func,delta_sigfun)[0] > 0:
    pl.plot(mysamp,myy[i],"ro")
else:
    pl.plot(mysamp,myy[i],"g*")
i+=1
for xn in xrange(0,len(x)):
    if simulate(x[xn],sigmoid_func,delta_sigfun)[0] > 0:
        pl.plot(train_x[xn][0],train_x[xn][1],"bo")
    else:
        pl.plot(train_x[xn][0],train_x[xn][1],"b*")
pl.show()

```

从如图 7-17 所示的分类效果中，能直观感受到多层感知器的非线性分类能力。其中，星号与实心圆圈表示的数据点被分成了两类，它们之间的界限是非线性表示的曲线，而不是线性表示的直线。

在多层感知器学习中，经常需要用到一个概念：误差曲线。误差曲线体现在一个二维坐标系中， X 轴表示训练次数， Y 轴表示每次训练的误差率。理想的误差曲线是随着 X 的增加， Y 值不断平滑减少，这与误差曲面相似，误差曲面以参数为自变量，这些都形象地体现了梯度下降的概念。它们都说明了一个道理：一个设计良好的多层次感知器，随着训练次数的增多，模型会越来越精确，误差曲面会朝着最小点向下滑动，而且误差曲线也在下降。

前面为了讲述的需要，“多层感知器神经网络源代码.doc”文件所示代码对误差率的计算以最终输出结果为依据计算，现在在神经网络输出层的后面再增加一层最终输出层，对原输出层的结果进行再次加工后输出。本书在以后涉及的相关代码中，如未特别说明，最终输出结果是指新增的最终输出层的输出结果。在此，将误差率的计算公式定义为：

$$\text{误差率} = \sqrt{\frac{\sum (\text{实际值} - \text{输出值})^2}{\text{样本数目}}}$$

设置好网络的相关参数，将期望误差率设置为 0.3，然后加入绘制误差曲线的代码。在列出代码之前，先介绍一下前面没有具体涉及的问题：中间隐藏层、学习率及动量参数。

一般认为，增加隐藏层数可以降低网络误差，提高精度，但也可使网络复杂化。当然也有学者提出了反对意见，但从实践经验来看，隐藏层的数目过少无法实现对复杂数据的学习，隐藏层的增多增加了网络的训练时间和出现“过拟合”的机率。

设计神经网络至少应考虑 3 层网络，其中有一层隐藏层，在这里将隐藏层的层数及每层节点数定义为如下形式：

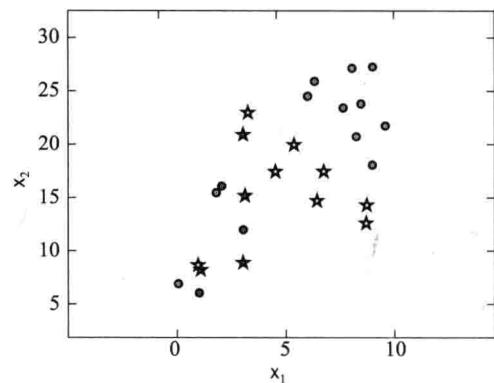


图 7-17 非线性分类

隐藏层的层数 = 每个样本的元素个数 × 4-2

每个隐藏层的节点数 = 训练样本数 - 1

目前仍有学者在对学习率和动量参数进行研究。什么样的学习率能既加快神经网络的训练时间，同时又能提高训练精度呢？动量参数设置为多少，更适合当前学习率，让误差曲线在下降过程中尽可能地跳出局部最小值的陷阱呢？

如果学习率设得过大，可能造成训练过早停止，错过了误差曲线的全局最小值；而设得过小则会造成训练时间加长，同时容易陷入误差曲线的局部最小值。动量参数也存在类似的问题。笔者认为这些参数需要在实践应用中进行调试和测试，通过对训练效果反复对比，才能得到最适合的参数。

根据以上设计思路，用 Python 在“多层感知器神经网络源代码.doc”文件代码的基础上实现。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-12.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy
isdebug=False
#x和d样本初始化
train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
warray_txn=len(train_x[0])
warray_n=len(train_x)-1
#基本参数初始化
oldmse=10**100
fh=1
maxtrycount=500
mycount=0.0
if maxtrycount>=20:
    r=maxtrycount/10
else:
    r=maxtrycount/2
#sigmoid函数
ann_sigfun=None
ann_delta_sigfun=None
#总层数初始化，比非线性导数多一层线性层
alllevel_count=warray_txn*4
# 非线性层数初始化
hidelevel_count=alllevel_count-1
#学习率参数
learn_r0=0.02
learn_r0*=2.5
learn_r=learn_r0
#动量参数
```

```

train_a0=learn_r0*1.2
train_a0*=0.0015
train_a=train_a0
#误差率
expect_e=0.3
.....
.....
x_max=len(err)
x_min=1
y_max=max(err)+0.2
y_min=0.
plt.xlabel(u"traincount")
plt.xlim(x_min, x_max)
plt.ylabel(u"mse")
plt.ylim(y_min, y_max)
lp_x1 = xrange(1, len(err)+1)
lp_x2 = err
plt.plot(lp_x1,lp_x2,'g-')
plt.show()

```

运行代码，执行结果如下：

```

.....
.....
-----开始第137次训练----- # # # # # 误差为: 0.309623
-----开始第138次训练----- # # # # # 误差为: 0.330235
-----开始第139次训练----- # # # # # 误差为: 0.284128
训练成功，正在进行检验
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
训练成功，输出误差为: 0.000000
样本: 4~11 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 7~340 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 10~95 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 3~29 =>

```

```

仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 7~43 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 5~128 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
测试值: 8.000000~70.000000
仿真计算中
[ 1.  0.]
正确目标值:[1,0]
测试值: 6.500000~272.000000
仿真计算中
[ 0.  1.]
正确目标值:[0,1]

```

从上述代码执行结果中不难看出，
网络在 139 次训练后，达到了训练误
差的期望值 0.3，对测试数据和样本数
据的验证均成功。再来看看如图 7-18 所示的误差曲线，总体呈现下滑趋势。

图 7-18 中的曲线并不平滑，如果将学习率设得更小，曲线将平滑很多，训练次数也将增多。现在修改代码，将学习率减少，重新绘制误差曲线图。

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-13.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy
isdebug=False
#x和d样本初始化
train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
warray_tnx=len(train_x[0])
warray_n=len(train_x)-1
#基本参数初始化
oldmse=10**100
fh=1
maxtrycount=2000
mycount=0.0

```

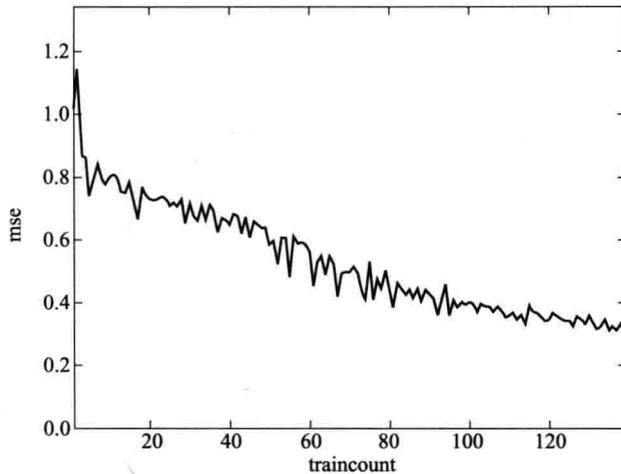


图 7-18 误差曲线（附彩图）

```

if maxtrycount>=20:
r=maxtrycount/10
else:
r=maxtrycount/2
#sigmoid函数
ann_sigfun=None
ann_delta_sigfun=None
#总层数初始化，比非线性导数多一层线性层
alllevel_count=warray_txn*4
# 非线性层数初始化
hidelevel_count=alllevel_count-1
#学习率参数
learn_r0=0.005
learn_r0*=2.5
learn_r=learn_r0
.....
.....

```

程序执行结果如下：

```

.....
.....
-----开始第354次训练----- # # # # # 误差为: 0.344559
-----开始第355次训练----- # # # # # 误差为: 0.347000
-----开始第356次训练----- # # # # # 误差为: 0.338863
-----开始第357次训练----- # # # # # 误差为: 0.342834
-----开始第358次训练----- # # # # # 误差为: 0.353655
-----开始第359次训练----- # # # # # 误差为: 0.280097
训练成功，正在进行检验
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
训练成功，输出误差为: 0.000000
样本: 4~11 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 7~340 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 10~5 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]

```

```

样本: 3==>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
样本: 7~43 =>
仿真计算中
[ 1.  0.]
=====正确目标值=====
[1, 0]
样本: 5~128 =>
仿真计算中
[ 0.  1.]
=====正确目标值=====
[0, 1]
测试值: 8.000000~70.000000
仿真计算中
[ 1.  0.]
正确目标值:[1,0]
测试值: 6.500000~272.000000
仿真计算中
[ 0.  1.]
正确目标值:[0,1]

```

从执行结果可看出，如果将学习率设得更小，训练次数确实增多了，由 100 多次变成了 300 多次，相比上次的训练，每次训练的误差率降低幅度变小，如图 7-19 所示为误差曲线。

相比图 7-18 的误差曲线，图 7-19 所示的误差曲线下降趋势更加明显，平滑很多，上下摆动的幅度也减小了不少。

7.1.4 Python 神经网络库

前面几节设计和完善了线性和非线性神经网络，并通过 Python 实现了神经网络，逐步阐述了怎样设计一个神经网络，如何改进神经网络，如何应用神经网络完成分类等。后面的章节还将对这些程序代码进行改进，介绍如何应用神经网络进行数据拟合。

“不用重复造轮子”，实践中除非条件限制（比如：受限嵌入式环境等应用）必须编写所有的神经网络代码，否则可以直接调用神经网络相关的库。

目前 Python 关于神经网络的库较多，这里选择纯 Python 库实现的神经网络库 Neurolab，在第 2 章中已经介绍过它的安装与配置方法，在此不再介绍，下面直接讲述如何应用它。首先进入 Python 的控制台，以前几节的数据为例，熟悉一下 Neurolab 的基本使用方法。

```
>>> import neurolab as nl
```

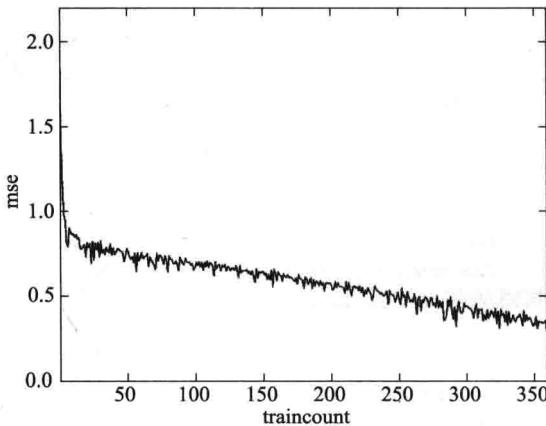


图 7-19 误差曲线

```

>>> train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
>>> input=np.array(train_x)
>>> d =[[1],[0],[1],[0],[1],[0]]
>>> target=np.array(d)
>>> target
array([[1],
       [0],
       [1],
       [0],
       [1],
       [0]])
>>> input
array([[ 4,  11],
       [ 7, 340],
       [10,  95],
       [ 3,  29],
       [ 7,  43],
       [ 5, 128]])
>>> net = nl.net.newff([[3, 10], [11, 400]], [5, 1])
>>> err = net.train(input, target, show=15)
Epoch: 15; Error: 0.326594518922;
Epoch: 30; Error: 0.0242485565317;
The goal of learning is reached

```

训练完毕后，再用数据测试该网络。代码如下：

```

>>> net.sim([[3, 10]])
array([[ 0.99999326]])
>>> net.sim([[9, 80]])
array([[ 0.89054486]])
>>> net.sim([[6.5,272]])
array([[ 0.05707987]])
>>> net.sim([[10,80]])
array([[ 0.9448553]])
>>> net.sim([[5,125]])
array([[ 0.12198103]])
>>> net.sim([[5,100]])
array([[ 0.18880761]])

```

上述代码显示，训练很成功，测试数据也被正确分类。

拥有中间隐藏层的多层感知器在输入与输出之间建立了映射关系，这种映射关系不一定是某种数学模型明确定义的。如果随机生成无规律的输入值和输出值，这些值之前的关系完全未知，那么是不能建立拥有确定数学公式的映射的，但可通过神经网络来建立它们之间的数学模型。代码如下：

```

>>> import numpy as np
>>> import neurolab as nl
>>> input = np.random.uniform(-0.5, 0.5, (10, 2))
>>> output = np.random.uniform(-0.5, 0.5, (10, 1))
>>> err = net.train(input, output, show=15)
>>> net = nl.net.newff([-0.5, 0.5], [-0.5, 0.5]), [8, 1])

```

```
>>> err = net.train(input, output, show=15)
Epoch: 15; Error: 0.0676764691815;
Epoch: 30; Error: 0.0131047452439;
The goal of learning is reached
>>> err[len(err)-1]
0.0097660775986454906
```

上述代码中的 `err[len(err)-1]` 表示训练停止后的误差率。可以看到，训练后精度很高，误差率小于 0.01。下面来编写代码绘制误差曲线。

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(err)
[<matplotlib.lines.Line2D object at
0x04C173B0>]
```

如图 7-20 所示是生成的误差曲线，能明显看到误差曲线很平滑。但此处的误差来源数据是训练 15 次采集一次，前几节中的误差曲线是每训练一次采集一次。因此，图 7-20 比前几节的误差曲线更为平滑。

通过多层感知器神经网络，在这组没有联系的随机生成的输入和输出之间“强行”建立了一种映射关系。

在测试时，通过未在样本中出现的输入数据，能得到符合这种映射关系的输出。这种映射能力非常重要，是后面几章提到的数据拟合、模式识别等机器学习任务的基础。

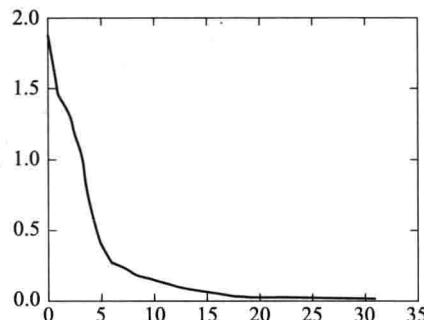


图 7-20 误差曲线

7.2 统计算法

统计分析算法在机器学习中占有重要地位，在此，仅介绍几种常用算法。

7.2.1 平均值

平均值是统计学中最常用的统计量，用来表明资料中各观测值集中较多的中心位置，用于反映现象总体的一般水平，或分布的集中趋势。有限总体的平均值定义为：

$$\mu = \frac{1}{N} \sum_{i=1}^n x_i$$

式中， μ 表示总体平均值， N 表示总体所包含的个体数。

平均值的意义在于：不仅可用它来反映一组数据的一般情况，还可以进行不同数据组之间的比较，以看出组与组之间的差别。下面随机生成两组随机数，对它们进行分析。代码如下：

```
>>> y = np.random.uniform(-0.5, 0.5, (10, 1))
>>> x = np.random.uniform(-0.5, 0.5, (10, 1))
>>> x
array([[-0.46954884],
       [-0.39078561],
```

```

[ 0.05531789],
[-0.06414516],
[-0.00511995],
[-0.41105398],
[-0.23416933],
[ 0.1369419 ],
[-0.45076555],
[ 0.1808625 ])
>>> y
array([[ 0.38100971],
       [ 0.12510564],
       [ 0.0540655 ],
       [-0.25050503],
       [ 0.13180995],
       [ 0.32953768],
       [-0.35940215],
       [-0.00294618],
       [-0.23359633],
       [-0.49808591]])
>>> np.mean(y)
-0.032300713545130311
>>> np.mean(x)
-0.16524661414915703

```

观察上述代码的执行结果可以看到， y 的平均值为 -0.032300713545130311 ， x 的平均值为 -0.16524661414915703 ， y 的平均值更趋向于 0，因此，我们预言： y 的总体分布要比 x 偏右，更趋向于 0。继续在 Python 控制台中绘制它们在坐标系中的分布，以验证这个预言。代码如下：

```

>>> z=np.zeros(10)
>>> z
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> xx=np.array(zip(z,x))
>>> xx
array([[ 0.          , -0.46954884],
       [ 0.          , -0.39078561],
       [ 0.          ,  0.05531789],
       [ 0.          , -0.06414516],
       [ 0.          , -0.00511995],
       [ 0.          , -0.41105398],
       [ 0.          , -0.23416933],
       [ 0.          ,  0.1369419 ],
       [ 0.          , -0.45076555],
       [ 0.          ,  0.1808625 ]])
>>> import matplotlib.pyplot as plt
>>> plt.xlim(-0.5, 0.5)
(-0.5, 0.5)
>>> plt.ylim(-0.01, 0.01)
(-0.01, 0.01)
>>> plot(yy[:,0],yy[:,1],'or')

```

```
[<matplotlib.lines.Line2D object at 0x04EB44D0>
>>> plot(xx[:,0],xx[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04CE2810>
>>>
```

x 用星号表示, y 用实心圆圈表示, 绘制图形如图 7-21 所示。从图 7-21 所示的效果图可以看出, 星号表示的数据比实心圆圈表示的数据整体更偏左, 更偏向 X 轴的负方向, 这个趋势证实了我们刚才的预言。

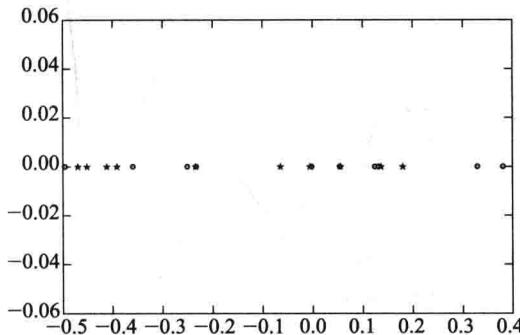


图 7-21 数据点分布

7.2.2 方差与标准差

1. 标准差

标准差是一组数据平均值分散程度的一种度量方式, 其计算公式为:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

其中, μ 为平均值。

较大的标准差, 代表大部分数值和其平均值之间差异较大; 较小的标准差, 代表这些数值较接近平均值。

2. 方差

标准差的平方就是方差, 其计算公式为:

$$\sigma^2 = \frac{\sum_{i=1}^N (x_i - \mu)^2}{N}$$

其中, μ 为平均值。

方差与标准差的统计意义差不多, 都是反映数据平均值分散程度。

3. Python 实现

下面用 Python 来实现标准差和方差的计算。在 Python 控制台操作, 随机生成一组 x 和 y 值, y 是均匀分布, x 是高斯分布。代码如下:

```
>>> y = np.random.rand(100)
```

```
>>> x = np.random.normal(0.5, 0.00001, 100)
>>> import matplotlib.pyplot as plt
```

观察 x 的分布趋势，如下所示：

```
>>> z=np.zeros(100)
>>> xx=np.array(zip(x,z))
>>> plot(xx[:,0],xx[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04E06BF0>]
```

x 的分布如图 7-22 所示，数据点在 X 轴上分布不均匀，主要点集中在 0.5 附近（图 7-22 的基值是 $4.9997e-1$ ），少量点散布在两边，这是预料之中的。

x 是高斯分布，随机生成 x 时，指定了参数为：平均值 0.5，最大标准差为 0.00001。高斯分布就是正态分布，具有集中性的特点，即：正态曲线的高峰位于正中央，即平均值所在的位置，如果是一维空间，则集中在 X 轴中平均值附近。

观察 y 的分布趋势，如下所示：

```
>>> z=np.zeros(100)
>>> yy=np.array(zip(y,z))
>>> plot(yy[:,0],yy[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04E06BF0>]
```

从分布图 7-23 来看， y 均匀分布在 X 轴上，没出现大量数据点聚集的情况。

现在分别求出 x 和 y 的平均值、方差与标准差。

```
>>> np.mean(x) #平均值
0.5000006367240246
>>> np.var(x) #方差
9.791184624177845e-11
>>> np.std(x) #标准差
9.8950414977289737e-06
>>> np.mean(y) #平均值
0.52212510115195176
>>> np.std(y) #标准差
0.28755684683792165
>>> np.var(y) #方差
0.082688940163367933
```

y 的平均值与 x 接近，但其方差和标准差不一样， x 的方差和标准差都比 y 小很多，比 x 小很多，这说明 x 相对于 y 来说，有更多的数值集中在平均值 0.5 附近，这就是方差和标准差的统计意义。

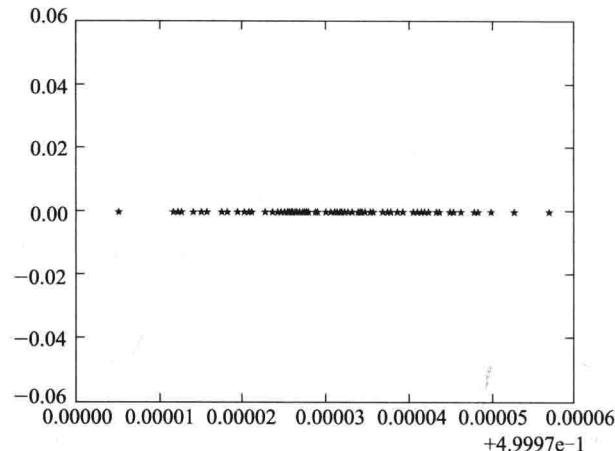


图 7-22 数据点分布

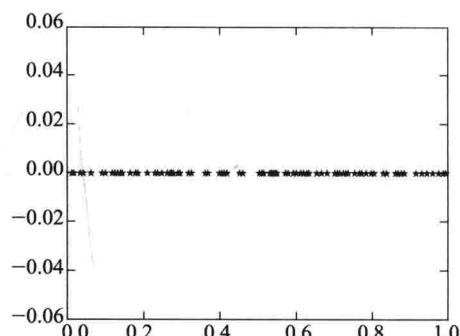


图 7-23 数据点分布（附彩图）

7.2.3 贝叶斯算法

1. 贝叶斯定理

贝叶斯定理是关于随机事件 A 和 B 条件概率的一则定理，下式定义了在事件 B 发生的情况下事件 A 发生的条件概率：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

假设 $\{A_i\}$ 是事件集合里的部分集合，其中， A_1, A_2, \dots, A_n 是某个过程中若干可能的前提，则 $P(A_i)$ 是对各前提条件出现可能性的事先估计，称之为先验概率。如果在这个过程得到了结果 B ，贝叶斯公式定义了 $P(A_i|B)$ ，它是对以 B 为前提下 A_i 出现概率的估计，则称 $P(A_i|B)$ 为后验概率。

对于任意的 A_i ，贝叶斯定理用下式来表示：

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

下面是一个贝叶斯算法的经典例子。

已知某种疾病的发病率是 0.001，即 1000 人中会有 1 个人得病，现有一种试剂可以检验患者是否得病，它的准确率是 0.99，即在患者确实得病的情况下，它有 99% 的可能呈现阳性。它的误报率是 5%，即在患者没有得病的情况下，它有 5% 的可能呈现阳性。现有一个病人的检验结果为阳性，请问他确实得病的可能性有多大？

假定 A 事件表示得病，那么 $P(A)$ 为 0.001，这就是“先验概率”，即没有做试验之前预计的发病率；假定 B 事件表示阳性，那么要计算的就是 $P(A|B)$ ，即“后验概率”，表示做了试验以后对发病率的估计。计算过程如下：

$$P(A|B) = P(A) \frac{P(B|A)}{P(B|A)P(A) + P(B|\bar{A})P(\bar{A})}$$

$$P(A|B) = 0.01 \times \frac{0.99}{0.99 \times 0.001 + 0.05 \times 0.999} \approx 0.019$$

通过计算，得知 $P(A|B)$ 约等于 0.019，即使检验呈现阳性，病人得病的概率也只是不到 2%，也就是说，呈现“假阳性”，阳性结果完全不足以说明病人得病。

贝叶斯算法在机器学习中主要用于分类，其基本原理是：已知样本 X ，首先计算 $P(X|C_i)$ ，得出 C_i 类别包含样本 X 的先验概率；然后根据贝叶斯定理求后验概率 $P(C_i|X)$ ，得到 X 属于 C_i 类别的后验概率；最后根据最大后验概率判断所属类别。

2. 朴素贝叶斯

贝叶斯算法的基础是概率推理，是在各种条件的存在不确定、仅知其出现概率的情况下，完成推理和决策任务。而朴素贝叶斯算法是基于独立假设的，即假设样本每个特征与其他特征都不相关。

尽管实际上独立假设常常是不准确的，但朴素贝叶斯分类器的若干特性让其在实践中能

够取得令人惊奇的效果，各类条件特征之间的解耦意味着每个特征的分布都可以独立地被当作一维分布来估计，减轻了由于维数灾带来的阻碍，避免了样本规模呈指数增长。在本书的第 10 章中有关于朴素贝叶斯算法的应用实例。

朴素贝叶斯算法通常在自动分类中使用，算法的主要过程为：首先，计算待分类样本特征的后验概率，然后使用最大似然比贝叶斯分类法或最小风险贝叶斯分类法完成分类。

当待分类样本拥有若干特征变量 F_1, F_2, \dots, F_n 时，该待分类样本属于类 C 的后验概率为：

$$p(C | F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^n p(F_i | C)$$

其中，证据因子 Z 是一个仅依赖 F_1, \dots, F_n 的缩放因子，当特征变量的值已知时是一个常数。

在朴素贝叶斯算法中，后验概率仅依赖于两个因素：类先验概率 $p(C)$ 和基于独立假设的先验概率 $p(F_i | C)$ 。

上面的推导结果解决了待分类样本后验概率的计算问题，余下的问题是：既然得到了属于每个类别的后验概率，如何知道待分类样本究竟属于哪个类别呢？有两个方法：使用最大似然比贝叶斯分类法和最小风险贝叶斯分类法。

目前普遍使用的是最大似然比贝叶斯分类方法，即：选出最有可能的那个，将待分类样本划归到后验概率最大的那一类中，也称为最大后验概率（MAP）决策准则。当采取最大后验概率决策时，相应的分类器公式定义如下：

$$\text{classify}(f_1, \dots, f_n) = \arg \max_c p(C=c) \prod_{i=1}^n p(F_i = f_i | C=c)$$

为了便于理解，去除这些繁杂的数学符号，将最终的分类器定义为：

样本所属类别 = $\arg \max(P(\text{类别}) \times \text{样本每个单独属性先验概率的累乘})$

其中， $\arg\max$ 表示选择最大概率的类别为最终类别， $P(\text{类别})$ 为类别先验概率，是指该类别本身的可能性有多少。

朴素贝叶斯虽然是基于独立假设的，但实践证明这种方法确实很有效。创建了 Viaweb (1998 年，Viaweb 成为最流行的电子商务软件，被雅虎收购，改称雅虎商店) 的美国著名程序员 Paul Graham 提出使用朴素贝叶斯识别垃圾邮件的方法，它通过使用字词等标记与垃圾邮件、非垃圾邮件的关联，计算一封邮件为垃圾邮件的可能性，以实现对垃圾邮件的判别。有兴趣的读者可以查阅下面网址：

<http://www.paulgraham.com/spam.html>

一个有趣的例子如下：假设有一台智能机器，负责将一堆未知水果分为 3 类：苹果、桂圆、香蕉。已知，这一堆水果中，苹果的数量约占 60%，桂圆的数量约占 35%，香蕉的数量约占 5%。

现在需要一位工程师为这台机器设计一个机器学习算法，这位工程师选择了朴素贝叶斯分类算法，将算法过程定义为如下形式：

首先，计算 $P(\text{类别})$ 。

$$P(\text{苹果})=0.6, P(\text{桂圆})=0.35, P(\text{香蕉})=0.05$$

然后，分别准备几个苹果、桂圆、香蕉的样本，分析其重量、颜色、形状，得出它们的先验概率。接着，重头戏开始了，识别未知水果。机器面前摆了一个这样的水果（用人类的“智能”，识别出这是一个香蕉，但机器识别出来不容易，目前人工智能水平远没达到期望的水平），如图 7-24 所示。

机器要做的识别过程是：

(1) 提取这个未知样本的特征：重量、颜色、形状。重量测量使用称重仪器，颜色靠色敏元件，形状使用图像识别算法。

(2) 从存储器中取出事先计算好的重量、颜色、形状的先验概率，设这个未知水果的重量、颜色、形状特征值为 a_1 、 a_2 、 a_3 ，它们各自的先验概率为：

$$P(a_1 \mid \text{苹果}) = 0.6, P(a_2 \mid \text{苹果}) = 0.4, P(a_3 \mid \text{苹果}) = 0.001$$

$$P(a_1 \mid \text{桂圆}) = 0.001, P(a_2 \mid \text{桂圆}) = 0.9, P(a_3 \mid \text{桂圆}) = 0.001$$

$$P(a_1 \mid \text{香蕉}) = 0.6, P(a_2 \mid \text{香蕉}) = 0.9, P(a_3 \mid \text{香蕉}) = 0.95$$

其中， a_1 =重量， a_2 =颜色， a_3 =形状。

(3) 计算后验概率。

设这个未知水果为 x ，计算 x 属于每个类别的概率。

$$\begin{aligned} P(\text{苹果} | x) &= P(\text{苹果} | a_1, a_2, a_3) \\ &= P(\text{苹果}) * (P(a_1 | \text{苹果}) * P(a_2 | \text{苹果}) * P(a_3 | \text{苹果})) \\ &= 0.6 * (0.6 * 0.4 * 0.001) \\ &= 0.6 * 0.00024 \\ &= 0.000144 \end{aligned}$$

$$\begin{aligned} P(\text{桂圆} | x) &= P(\text{桂圆} | a_1, a_2, a_3) \\ &= P(\text{桂圆}) * (P(a_1 | \text{桂圆}) * P(a_2 | \text{桂圆}) * P(a_3 | \text{桂圆})) \\ &= 0.35 * (0.001 * 0.9 * 0.001) \\ &= 0.000000315 \end{aligned}$$

$$\begin{aligned} P(\text{香蕉} | x) &= P(\text{香蕉} | a_1, a_2, a_3) \\ &= P(\text{香蕉}) * (P(a_1 | \text{香蕉}) * P(a_2 | \text{香蕉}) * P(a_3 | \text{香蕉})) \\ &= 0.05 * (0.6 * 0.9 * 0.95) \\ &= 0.02565 \end{aligned}$$

(4) 寻找最大后验概率得到所属类别，假设所属类别为 C 。

$$\begin{aligned} P(C|x) &= \operatorname{argmax}(P(\text{苹果} | x), P(\text{桂圆} | x), P(\text{香蕉} | x)) \\ &= \operatorname{argmax}(0.000144, 0.000000315, 0.02565) \end{aligned}$$

3 个概率中，0.02565 是最大的概率，因此 $C=\text{香蕉}$ 。

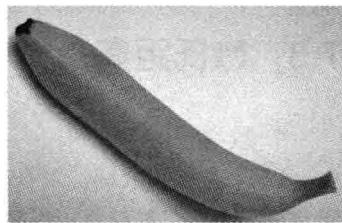


图 7-24 香蕉

(5) 将这个水果分到香蕉堆中。

7.3 欧氏距离

1. 数学原理

以 R 表示实数域。对任意一个正整数 n , 实数的 n 元组的全体构成了 R 上的一个 n 维向量空间, 用 R^n 来表示, 也称之为实数坐标空间。 R^n 中的元素写作 $X=(x_1, x_2, \dots, x_n)$, 这里的 x_i 都是实数。

欧氏范数定义 R^n 上的距离函数 (或称度量) :

$$d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

这个距离函数称为欧几里得度量, 也称欧氏距离。欧氏距离通常用于衡量两个点之间的距离, 这两个点可以是定义在二维空间的, 也可以是定义在三维空间或者 n 维空间的。

2. 计算实例

假设在二维空间中, 已定义两个点: (3,8) 和 (2,5), 其欧氏距离如下:

$$d = \sqrt{(3-2)^2 + (8-5)^2} = 3.1623$$

经过计算, 该距离为 3.1623, 两点在空间的表示如图 7-25 所示。该距离为直线的长度。

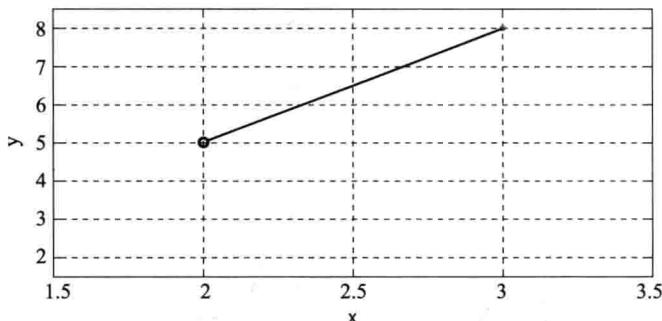


图 7-25 二维空间中两点的距离

假设有两点: (2,5,7) 和 (3,8,2),
它们在三维空间的欧氏距离定义为:

$$d = \sqrt{(3-2)^2 + (8-5)^2 + (7-2)^2} = 5.9161$$

经过计算, 该距离为 5.9161, 如图 7-26 所示。该距离为直线的长度。

n 维欧氏空间是一个点集, 它的每个点 X 可以表示为 $(x[1], x[2], \dots, x[n])$, 其中 $x[i](i = 1, 2, \dots, n)$ 是

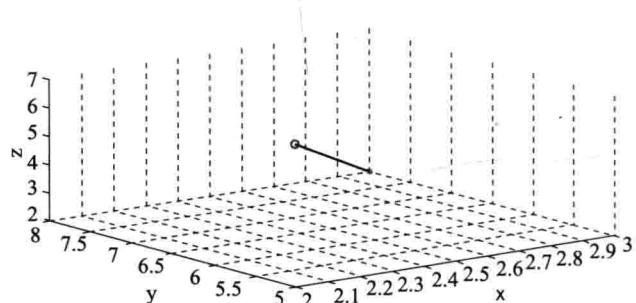


图 7-26 三维空间中两点的距离

实数，称为 X 的第 i 个坐标，两个点 $A=(a[1], a[2], \dots, a[n])$ 和 $B=(b[1], b[2], \dots, b[n])$ 之间的距离 $d(A, B)$ 计算方式如下：

$$d(A, B) = \sqrt{\sum (a[i] - b[i])^2}$$

7.4 余弦相似度

1. 数学原理

(1) 向量。空间中有两个点 A 和 B , $A \rightarrow B$ 就是一个向量，可以读成从 A 到 B 。它既有大小又有方向，设原点是 O ，给定空间中任意一点 C , OC 是从 O 到 C 的向量。如图 7-27 所示就是一个定义在三维空间上的向量 OA 。

(2) 点积(内积)。对任意两个向量 x, y , 点积 $\langle x, y \rangle$ ($x \cdot y$) 定义为：

$$\langle x, y \rangle = \sum_{i=1}^n x_i y_i = x_1 y_1 + x_2 y_2 + \dots + x_n y_n$$

R^n 中的两个向量通过点积的方式映射成一个实数值， R^n 及其点积称为 R^n 上的欧几里得结构，可以将 R^n 称为 n 维欧几里得空间，点积 $\langle \cdot, \cdot \rangle$ 称为欧氏点积。

(3) 向量长度。向量 X 的长度定义为：

$$\|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1}^n (x_i)^2}$$

上式又称为 R^n 上的欧氏范数。

(4) 余弦相似性。在欧氏内积和欧氏范数基础上定义向量 A 和 B 之间的 θ 余弦相似性如下：

$$\text{similarity} = \cos(\theta) = \frac{A \bullet B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

其中 θ 为 x 和 y 所夹的内角。

2. 算法原理

余弦相似性通过测量两个向量点积空间夹角的余弦值来度量它们之间的相似性。0 度角的余弦值是 1，而其他任何角度的余弦值都不大于 1，并且其最小值是 -1。

两个向量之间的角度余弦值确定两个向量是否大致指向相同的方向。两个向量有相同的指向时，余弦相似度的值为 1；两个向量夹角为 90° 时，余弦相似度的值为 0；两个向量指向完全相反的方向时，余弦相似度的值为 -1。

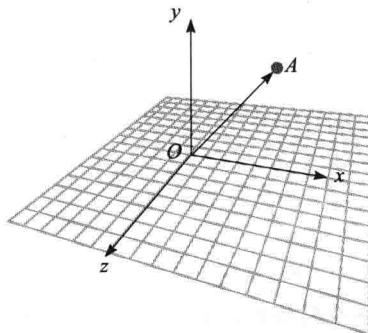


图 7-27 向量

余弦相似度通常用于两个向量的夹角在 90° 之内的情况，余弦相似度的值为 $0\sim 1$ 。余弦相似度可在任何维度的向量比较中，因此在高维空间中的应用非常广泛。

在实际应用中，通常会先提取两个数据的特征，每个数据各形成一个 n 维向量，然后通过计算这两个向量的余弦相似度来判定这两个向量是否是同一类型。例如：文本分类、图像识别等都能使用余弦相似度算法。

设定一个常数 C ($0 \leq C \leq 1$)， OA 与 OB 之间形成了夹角 α ，余弦相似度如果大于 C ，就认为 OA 与 OB 属于同一类，如图 7-28 所示。

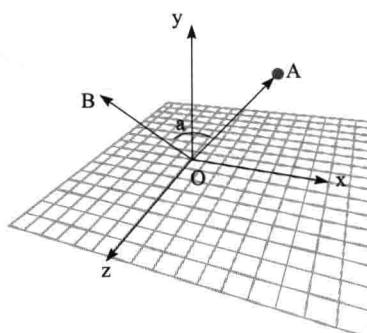


图 7-28 余弦相似度

7.5 SVM

1992~1995 年，Vapnik 等在 SLT 的基础上发展了 SVM 算法。它在解决小样本、非线性及高维模式识别问题中都表现出了许多特有的优势，目前已经成为与神经网络地位齐名的算法，在样本量较小的情况下，其实际运用效果甚至超过了神经网络。

7.5.1 数学原理

SVM 中文名为支持向量机，英文全称为 Support Vector Machine，是一种监督式学习的方法。它拥有坚实的数学基础，数学原理较复杂，涉及的数学知识很多，如果将其相关理论以及推导完全展开来，能写成一本超过 300 页的书。因此，这里仅简单介绍 SVM 的基本原理。

1. 算法策略

SVM 首先将向量映射到一个更高维的空间里，在其中建立最大间隔超平面，将数据分开；然后，在超平面两边再设立两个互相平行的超平面；最后分离超平面，使两个平行超平面的距离最大化。SVM 假定平行超平面间的距离或差距越大，分类器的总误差越小。

2. 超平面

超平面的数学形式可以写作：

$$w \cdot x - b = 0$$

其中 x 是超平面上的点， w 是垂直于超平面的向量。

由图 7-29 可知，平行超平面可表示为以下两个方程：

$$w \cdot x - b = 1$$

$$w \cdot x - b = -1$$

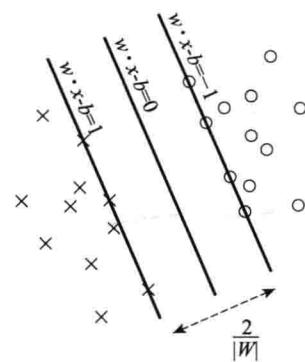


图 7-29 超平面

其中， w 为超平面的法向量，是一个变量。

如果数据是线性可分的，可找到两个超平面，在它们之间没有任何样本点，并且这两个超平面之间的距离也最大。

这两个超平面之间的距离是 $2/\|w\|$ ，因此需要最小化 $\|w\|$ ，因为这两个超平面之间没有任何样本点，所以 x_i 还需要满足以下两个条件中的一个：

$$w \cdot x_i - b \geq 1$$

$$w \cdot x_i - b \leq -1$$

把上面两个式子合写，将变成：

$$C_i(w \cdot x_i - b) \geq -1, \quad 1 \leq i \leq n$$

3. 二次规划最优化

二次规划问题可以用以下形式来描述。

(1) 函数 $f(x)$ 定义为：

$$f(x) = (1/2)x^T Qx + c^T x$$

其中， x^T 是 x 的转置。

(2) 函数受到一个或更多如下形式的限制条件：

$$Ax \leq b$$

$$Ex = d$$

如果 Q 是半正定矩阵，那么 $f(x)$ 是一个凸函数。如果有至少一个向量 x 满足约束而且 $f(x)$ 在可行域有下界，二次规划问题就有一个全局最小值 x 。如果 Q 是正定矩阵，那么全局最小值就是唯一的。如果 $Q=0$ ，二次规划问题就变成线性规划问题。

再来看 SVM 算法，该算法提出了两个要求：第一，超平面之间没有任何样本点；第二，超平面之间的距离最大。这样就形成了二次规划最优化问题。

$$\text{minimize: } W(a) = \frac{\|w\|^2}{2} + C \sum_i \xi_i$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

这也是一个二次凸规划问题。

根据优化理论，一个点 x 成为全局最小值的必要条件是满足 Karush-Kuhn-Tucker (KKT) 条件，当 $f(x)$ 是凸函数时，KKT 条件也是充分条件。因此通过拉格朗日变换以及 KKT 定理推导，二次凸规划最优化问题转变为：

$$\text{minimize: } W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0$$

然后引入核函数，最后的目标函数为：

$$\text{maximize: } W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to: } \sum_{i=1}^n \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, \forall i$$

改写为矩阵相乘的格式，得到：

$$\begin{aligned} \text{minimize: } f(\alpha) &= \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to } y^T \alpha &= 0 \quad 0 \leq \alpha_i \leq C, i=1, \dots, l \end{aligned}$$

最终的目标变成了：通过训练样本寻找 α ，使得下式最小：

$$\frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

核函数可以是线性的或非线性的，线性核函数仅能完成线性分类，而非线性核函数既可以完成线性分类，也可以完成非线性分类。

7.5.2 SMO 算法

SMO（最小贯序列方法）是改进的 SVM 训练算法，同其他 SVM 训练算法一样，SMO 会将一个大的 QP 问题分解为一系列小的 QP 问题。与其他算法不一样的是，SMO 处理的是规模最小的 QP 问题，因此能够快速解决并获得解析解，而且能够有效地改善空间和时间复杂度。

SMO 基本原理是：每次仅选择两个元素共同优化，在其他参数固定的前提下，找到这两个参数的最优值，并更新相应的 α 向量，而这两个点乘子的优化可以获得解析解。

7.5.3 算法应用

SVM 算法的数学原理较难懂，推导过程复杂，掌握 SVM 算法的关键在于动手实践。通过编写代码，运用 SVM 算法解决机器学习问题才能真正理解它。mlpy 库提供了 SVM 算法的相关函数，本节将以 mlpy 库为例进行阐述，请按第 2 章的指导安装 mlpy 库。

1. 核函数

mlpy 的 SVM 算法库提供以下核函数。

线性核函数通常表现为直线方程，函数名为 linear。

非线性核函数如下：

- (1) 多项式函数，函数名为 poly。
- (2) 径向基函数，函数名为 rbf。
- (3) sigmoid 函数，函数名为 sigmoid。

2. 线性分类

下面使用线性核作为 SVM 的核函数，对下面的数据进行分类。

```
x = [[1, 8], [3, 20], [1, 15], [3, 35], [5, 35], [4, 40], [7, 80], [6, 49]]
y = [1, 1, 0, 0, 1, 0, 0, 1]
```

创建 SVM 类的实例，并使用 learn 方法训练 SVM。

```
x=np.array(x)
y=np.array(y)
svm = mlpv.LibSvm()
svm.learn(x, y)
```

使用 pred 方法对未知数据进行分类。代码如下：

```
ty=svm.pred(tlp_x[ii])
```

完整的 Python 代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#email:myhaspl@qq.com
#author:麦好
#7-14.py
import numpy as np
import matplotlib.pyplot as plt
import mlpv
print 'loading ...'

x = [[1,8],[3,20],[1,15],[3,35],[5,35],[4,40],[7,80],[6,49]]
y=[1,1,0,0,1,0,0,1]
showpoint=['ro','bo']
tshowpoint=['r*','b*']
x=np.array(x)
y=np.array(y)
svm = mlpv.LibSvm()
svm.learn(x, y)
lp_x1 = x[:,0]
lp_x2 = x[:,1]
xmin, xmax = np.min(lp_x1)-1, np.max(lp_x1)+1
ymin, ymax = np.min(lp_x2)-1, np.max(lp_x2)+1
plt.subplot(111)
plt.xlabel(u"x")
plt.xlim(xmin, xmax)
plt.ylabel(u"y")
plt.ylim(ymin, ymax)
#显示样本点
for ii in xrange(0,len(x)):
    ty=svm.pred(x[ii])
    if ty>0:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])
    else:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])
#未知样本分类
tlp_x1=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x2=np.random.rand(50)*(ymax-ymin)+xmin
tlp_x=np.array(zip(tlp_x1, tlp_x2))
for ii in xrange(0,len(tlp_x)):
    ty=svm.pred(tlp_x[ii])
```

```

if ty>0:
    plt.plot(tlp_x1[ii], tlp_x2[ii], tshowpoint[int(ty)])
else:
    plt.plot(tlp_x1[ii], tlp_x2[ii], tshowpoint[int(ty)])
plt.show()

```

上述代码中，随机生成了 50 个数据作为待分类未知样本。如图 7-30 所示是程序生成的分类散点图，实心圆圈和星号分别代表两类数据。从图上能直观看到，分类很成功。

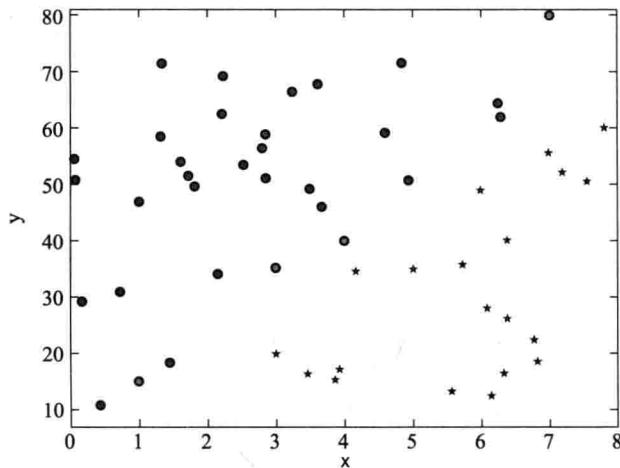


图 7-30 分类散点图

3. 非线性分类

非线性分类需要选择非线性核函数，这里选择 poly 作为 SVM 的核函数。尝试将分属于下面两个函数的坐标点分开：

第一类： $y=x^a+b$ ($a \leq 2$, $abs(b) < 10$)

第二类： $y=x^a+b$ ($a \geq 3$, $abs(b) < 10$)

(1) 设置样本数据。代码如下：

```

x= [[1,1],[2,4],[3,12],[9,70],[5,130],[4,13],[5,29],[5,135],[4,68],[10,1000],[8,
520],[7,340],[6,40],[10,150]]
y=[1,1,1,1,0,1,1,0,0,0,0,0,1,1]

```

(2) 设置 SVM 的核函数，进行训练。

```

x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=10)
svm.learn(x, y)

```

(3) 随机产生待分类数据，进行测试。代码如下：

```

tlp_x10=np.random.rand(100)*(xmax-xmin)+xmin
tlp_x20=tlp_x10**3+np.random.rand(100)*20-10
tlp_x11=np.random.rand(100)*(xmax-xmin)+xmin

```

```

tlp_x21=tlp_x11**2+np.random.rand(100)*20-10
tlp_x30=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x31=tlp_x30***(round(np.random.rand()*6,0)+3)+np.random.rand(50)*10-5
tlp_x40=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x41=tlp_x30***(round(np.random.rand(),0)+1)+np.random.rand(50)*10-5

```

如图 7-31 所示是程序生成的散点图，实心圆圈和星号分别代表不同类的数据。数据点被成功划分为两类：图的左上部呈现递增曲线趋势的数据点划分到了第 2 类函数，其余被划分到第 1 类。

完整的 Python 代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#email:myhaspl@qq.com
#author:麦好
#7-15.py
import numpy as np
import matplotlib.pyplot as plt
import mlpy

print 'loading ...'

x = [[1,1],[2,4],[3,12],[9,70],[5,130],[4,13],[5,29],[5,135],[4,65],[10,1000],[8,520],[7,340],[6,40],[10,150]]
y=[1,1,1,1,0,1,1,0,0,0,0,0,1,1]
showpoint=['ro','r*']
tshowpoint=['bo','b*']
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=50)
svm.learn(x, y)
lp_x1 = x[:,0]
lp_x2 = x[:,1]
xmin, xmax = np.min(lp_x1)-0.5, np.max(lp_x1)+0.5
ymin, ymax = np.min(lp_x2)-0.5, np.max(lp_x2)+0.5
plt.subplot(111)
plt.xlabel(u"x")
plt.xlim(xmin, xmax)
plt.ylabel(u"y")
plt.ylim(ymin, ymax)

#显示样本点
for ii in xrange(0,len(x)):
    ty=svm.pred(x[ii])
    if ty>0:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])
    else:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])

```

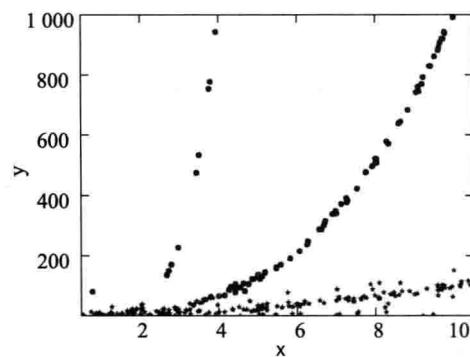


图 7-31 分类散点图

```

#未知样本分类
tlp_x10=np.random.rand(100)*(xmax-xmin)+xmin
tlp_x20=tlp_x10**3+np.random.rand(100)*20-10
tlp_x11=np.random.rand(100)*(xmax-xmin)+xmin
tlp_x21=tlp_x11**2+np.random.rand(100)*20-10
tlp_x30=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x31=tlp_x30**2*(round(np.random.rand()*6,0)+3)+np.random.rand(50)*10-5
tlp_x40=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x41=tlp_x30**2*(round(np.random.rand(),0)+1)+np.random.rand(50)*10-5

tlp_x1=tlp_x10.tolist()+tlp_x11.tolist()+tlp_x30.tolist()+tlp_x40.tolist()
tlp_x2=tlp_x20.tolist()+tlp_x21.tolist()+tlp_x31.tolist()+tlp_x41.tolist()

tlp_x=np.array(zip(tlp_x1,tlp_x2))
for ii in xrange(0,len(tlp_x)):
    ty=svm.predict(tlp_x[ii])
    if ty>0:
        plt.plot(tlp_x1[ii],tlp_x2[ii], tshowpoint[int(ty)])
    else:
        plt.plot(tlp_x1[ii],tlp_x2[ii], tshowpoint[int(ty)])
plt.show()

```

下面以 rbf 为核函数完成螺旋型空间下的分类。代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#7-16.py
import numpy as np
import matplotlib.pyplot as plt
import mlp
f = np.loadtxt("spiral.data")
x, y = f[:, :2], f[:, 2]
svm = mlp.LibSvm(svm_type='c_svc', kernel_type='rbf', gamma=100)
svm.learn(x, y)
xmin, xmax = x[:, 0].min()-0.1,
x[:, 0].max()+0.1
ymin, ymax = x[:, 1].min()-0.1,
x[:, 1].max()+0.1
xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.01), np.arange(ymin, ymax, 0.01))
xnew = np.c_[xx.ravel(), yy.ravel()]
ynew = svm.predict(xnew).reshape(xx.shape)
fig = plt.figure(1)
plt.pcolormesh(xx, yy, ynew)
plt.scatter(x[:, 0], x[:, 1], c=y)
plt.show()

```

如图 7-32 所示为分类的效果图。两类数据被螺旋型分界线划分。

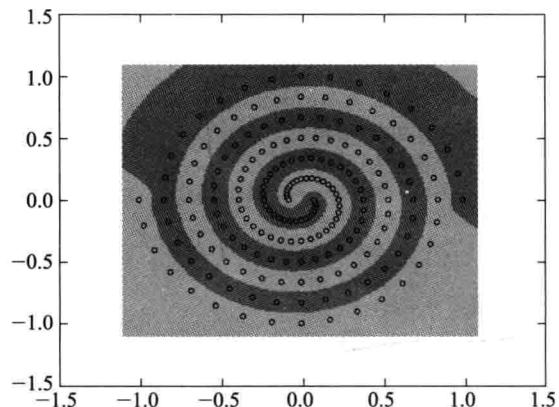


图 7-32 SVM 分类的效果图

7.6 回归算法

回归分析是统计学上分析数据的一种方法，目的在于了解两个或多个变量间是否相关，以及相关的方向与强度，并建立数学模型，以便观察特定变量来预测研究者感兴趣的变量，它建立了因变量与自变量之间的关系模型。

7.6.1 线性代数基础

线性代数是数学的一个分支，它的研究对象是向量、向量空间（或称线性空间）、线性变换和有限维的线性方程组。向量空间是现代数学的一个重要课题，线性代数广泛地应用于抽象代数和泛函分析中，并能通过解析几何具体表示。线性代数在机器学习理论体系中的地位举足轻重。

1. 伴随矩阵

在线性代数中，一个方形矩阵的伴随矩阵是一个类似于逆矩阵的概念。如果矩阵可逆，那么它的逆矩阵和它的伴随矩阵之间只差一个系数。

矩阵 A 的伴随矩阵是 A 的余子矩阵的转置矩阵，具体定义如下：

$$\text{设 } A=(a_{ij})_{n \times n}, \text{ 则 } a_{i1}A_{j1}+a_{i2}A_{j2}+\cdots+a_{in}A_{jn}=\begin{cases} |A|, & i=j \\ 0, & i \neq j \end{cases}$$

即 $AA^*=A^*A=|A|E$

其中，

$$A^* = \begin{bmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \dots & \dots & \dots & \dots \\ A_{1n} & A_{2n} & \dots & A_{nn} \end{bmatrix} = (A_{ji}) = (A_{ij})^T$$

若 A 可逆，则 A 的伴随矩阵 $A^*=|A|A^{-1}$, $(A^*)^*=\frac{1}{|A|}A$ 。

2. 方阵的行列式运算

设 A 、 B 为 n 阶方阵，则 $|AB|=|A||B|=|B||A|=|BA|$

但 $|A \pm B|=|A| \pm |B|$ 不一定成立。

3. 矩阵与方阵

$m \times n$ 个数 a_{ij} 排成 m 行 n 列的表格 $\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$ 称为矩阵，简记为 A ，或 $(a_{ij})_{m \times n}$ 。若

$m=n$ ，则称 A 是 n 阶矩阵或 n 阶方阵。

4. 矩阵的基本运算

设 $A=(a_{ij})$, $B=(b_{ij})$ 是两个 $m \times n$ 矩阵，则 $m \times n$ 矩阵 $C=(c_{ij})=a_{ij}+b_{ij}$ 称为矩阵 A 与 B 的和，

记为 $A+B=C$ 。

设 $A=(a_{ij})$ 是 $m \times n$ 矩阵, k 是一个常数, 则 $m \times n$ 矩阵 (ka_{ij}) 称为数 k 与矩阵 A 的数乘, 记为 kA 。

设 $A=(a_{ij})$ 是 $m \times n$ 矩阵, $B=(b_{ij})$ 是 $n \times s$ 矩阵, 那么 $m \times s$ 矩阵 $C=(c_{ij})$, 其中, $c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}b_{nj}=\sum_{k=1}^n a_{ik}b_{kj}$ 称为 A 与 B 的乘积, 记为 $C=AB$

5. 矩阵的秩

若 A 为 n 阶方阵, 则 $r(A^*)=\begin{cases} n, & r(A)=n \\ 1, & r(A)=n-1 \\ 0, & r(A) < n-1 \end{cases}$

6. 线性相关与线性无关

$\alpha_1, \alpha_2, \dots, \alpha_s$ 线性相关 \Leftrightarrow 至少有一个向量可以用其余向量线性表示。

若 $\alpha_1, \alpha_2, \dots, \alpha_s$ 线性无关, $\alpha_1, \alpha_2, \dots, \alpha_s, \beta$ 线性相关 $\Leftrightarrow \beta$ 可以由 $\alpha_1, \alpha_2, \dots, \alpha_s$ 唯一线性表示。

7.6.2 最小二乘法原理

1. 范数

在向量空间 $R^n(C^n)$ 中, 设 $x=(x_1, x_2, \dots, x_n)^T$, 向量范数定义如下:

x 的 2- 范数或欧氏范数计算公式为:

$$\|x\|_2 = (\|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_n\|^2)^{1/2}$$

x 的 1- 范数计算公式为:

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

x 的 ∞ 范数或最大范数计算公式为:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

x 的 p 范数计算公式为:

$$\|x\|_p = (\|x_1\|^p + \|x_2\|^p + \cdots + \|x_n\|^p)^{1/p}$$

A 的行范数计算公式如下:

$$\|A\|_\infty = \max_{\vec{x} \neq 0} \frac{\|A\vec{x}\|_\infty}{\|\vec{x}\|_\infty} = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$$

A 的列范数计算公式如下:

$$\|A\|_\infty = \max_{\vec{x} \neq 0} \frac{\|A\vec{x}\|_1}{\|\vec{x}\|_1} = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}|$$

A 的 2- 范数或谱范数计算公式如下:

$$\|A\|_2 = \max_{\vec{x} \neq 0} \frac{\|A\vec{x}\|_2}{\|\vec{x}\|_2} = \sqrt{\lambda_{\max}(A^T A)}$$

其中 $\lambda_{\max}(A^T A)$ 为 $A^T A$ 的最大特征值。

2. 算法目标

线性回归算法通过适合的参数，使函数与观测值之差的平方和最小，即：

$$\min_{\vec{x}} \sum_{i=1}^n (y_m - y_i)^2$$

(1) 二元线性回归。首先将线性回归函数定义为如下形式：

$$y = x_0 + x_1 t$$

根据其算法目标，得到下式：

$$\min_{x_0, x_1} \left\| \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_n \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right\|_2 \min_x \|Ax - b\|_2$$

解上式，得到如下形式：

$$x_1 = \frac{\sum_{i=1}^n (t_i - \bar{t})(y_i - \bar{y})}{\sum_{i=1}^n (t_i - \bar{t})^2}$$

$$x_0 = \bar{y} - x_1 \bar{t}$$

其中 $\bar{t} = \frac{1}{n} \sum_{i=1}^n t_i$ ，为 t 值的算术平均值。

(2) 多元线性回归。将多个不相关变量 t_1, \dots, t_q ，组成下面线性函数：

$$y(t_1, \dots, t_q; x_0, x_1, \dots, x_q) = x_0 + x_1 t_1 + \dots + x_q t_q$$

上式可写成 $Ax=b$ 的形式，如下所示：

$$\begin{pmatrix} 1 & t_{11} & \cdots & t_{1j} & \cdots & t_{1q} \\ 1 & t_{21} & \cdots & t_{2j} & \cdots & t_{2q} \\ \vdots & & & & & \\ 1 & t_{i1} & \cdots & t_{ij} & \cdots & t_{iq} \\ \vdots & & & & & \\ 1 & t_{n1} & \cdots & t_{nj} & \cdots & t_{nq} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_q \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}$$

接着求解目标函数：

$$\min_x \|Ax - b\|_2, A \in C^{m \times n}, b \in C^n$$

其特解为 A 的广义逆矩阵与 b 的乘积，这同时也是 2- 范数极小的解，其通解为特解加上 A 的零空间。

7.6.3 线性回归

1. 回归模型求解

上节讲述了一些多元线性回归模型，可表示为 $Ax=b$ 的形式，求解回归模型就是求解 A 代表的参数值的估计值，求解过程就是线性回归的过程。根据最小二乘原理及相关数学推导，参数估计值为：

$$\tilde{B} = (X'X)^{-1}X'T$$

2. 一元线性回归

如果随机变量 y 与变量 x 之间呈现某种线性关系，则 y 与 x 之间一元线性回归模型为：

$$\hat{y} = a + bx$$

上式也称变量 y 对变量 x 的一元线性回归方程， a 、 b 称为回归系数。

根据前面讲述的最小二乘法原理，用 Python 编码实现。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-17.py

import matplotlib.pyplot as plt
x =[1,2,3,3,6,12,11]
y =[3,5,8,5,12,26,20]
average_x=float(sum(x)//len(x))
average_y=float(sum(y))/len(y)
x_sub=map((lambda x:x-average_x),x)
y_sub=map((lambda x:x-average_y),y)
x_sub_pow2=map((lambda x:x**2),x_sub)
y_sub_pow2=map((lambda x:x**2),y_sub)
x_y=map((lambda x,y:x*y),x_sub,y_sub)
a=float(sum(x_y))/sum(x_sub_pow2)
b=average_y-a*average_x
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(x, y, '*')
plt.plot([0,15], [0*a+b,15*a+b])
plt.grid()
plt.title ("{} * x + {} ".format(a,b))
plt.show()
```

如图 7-33 所示为程序绘制的散点图，这些数据点呈明显的线性趋势，程序将它们的线性趋势绘制成一条回归线，回归效果不错。

3. 多元线性回归

一元线性回归是指将一个主要影响因素作为自变量，分析自变量的变化如何引起因变量的变化。在现实问题的研究中，因变量的变化往往受几个重要因素的影响，此时就需要用两个或两个以上的影响因素作为自变量

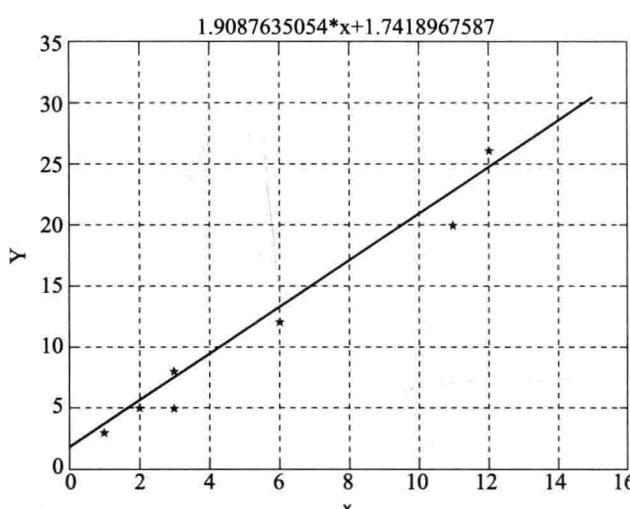


图 7-33 散点图

来解释因变量的变化，这就是多元回归。

设 y 为因变量， x_1, x_2, \dots, x_k 为自变量，并且自变量与因变量之间为线性关系时，则多元线性回归模型为：

$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_k x_k + e$$

根据最小二乘法，使用 Python 来实现这个回归模型。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-18.py
import numpy as np
x = np.matrix([[7, 2, 3], [3, 7, 17], [11, 3, 5]], dtype=np.float64)
y = np.matrix([28, 40, 44], dtype=np.float64).T
b=(x.T*x).I*x.T*y
print u"参数项矩阵为{0}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i, 0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye, mye, mye]).T
print "y=%f*x1+%f*x2+%f*x3+%f"%(cb[0], cb[1], cb[2], mye)
```

上述代码计算了参数估计值，并列出了多元线性回归方程。

```
参数项矩阵为[[ 3.]
 [ 2.]
 [ 1.]]
y=3.000000*x1+2.000000*x2+1.000000*x3+-0.000000
```

7.6.4 多元非线性回归

如果自变量 X_1, X_2, \dots, X_m 与因变量 Y 皆具有非线性关系，或者有的为非线性，有的为线性，则需要选择多元非线性回归模型。非线性回归方程很难求解，通常把非线性回归转化为线性回归，然后应用最小二乘法求解。

例如，非线性回归方程模型为：

$$\frac{1}{y} = a + \frac{b}{x}$$

首先，进行变量变换。

$$X = \frac{1}{x}$$

$$Y = \frac{1}{y}$$

经过变换后，将其化为线性模型。

$$Y = a + bX$$

然后，再用最小二乘法求出参数的估计值。最后经过适当的变换，得到所求回归曲线。

1. 一元三次回归模型

一元三次回归模为：

$$\hat{y}_3 = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$

用 Python 代码实现：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-19.py
#y=b1*x+b2*(x^2)+b3*(x^3)
import numpy as np
import matplotlib.pyplot as plt
z=np.matrix([3,1.4,1.9]).T
myx =np.matrix([[7],[3],[9]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,0]**2,myx[0,0]**3],\
               [myx[1,0],myx[1,0]**2,myx[1,0]**3],\
               [myx[2,0],myx[2,0]**2,myx[2,0]**3]],\
               dtype=np.float64)
y =x*z
b=(x.T*x).I*x.T*y
print u"参数矩阵为{0}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T
print "y=%f*x+%f*x^2+%f*x^3+%f"% (cb[0],cb[1],cb[2],mye)
```

```
pltx=np.linspace(0,10,1000)
plty=cb[0]*pltx+cb[1]*(pltx**2)+cb[2]*(pltx**3)+mye
plt.plot(myx,y,"*")
plt.plot(pltx,plty)
plt.show()
```

程序运行后，计算参数估计值以及最终的回归方程。

```
参数矩阵为[[ 3. ]
 [ 1.4]
 [ 1.9]]
y=3.000000*x+1.400000*x^2+1.900000*x^3+
0.000000
```

非线性回归方程表现为曲线，如图 7-34 所

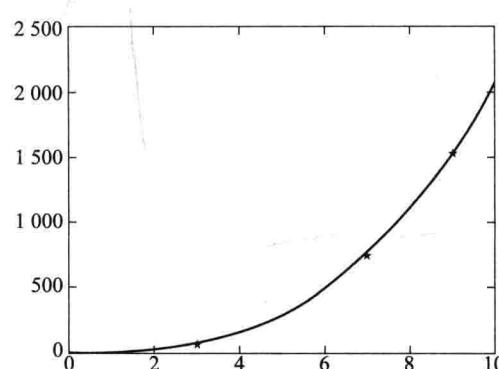


图 7-34 回归曲线

示是上述代码生成的回归曲线。

2. 二元二次多项式回归方程

二元二次多项式回归方程为：

$$\hat{y} = a + b_{11}x_1 + b_{21}x_2 + b_{12}x_1^2 + b_{22}x_2^2 + b_{11 \times 22}x_1x_2$$

令 $b_1 = b_{11}$, $b_2 = b_{21}$, $b_3 = b_{12}$, $b_4 = b_{22}$, $b_5 = b_{11 \times 22}$
 $x_3 = x_1^2$, $x_4 = x_2^2$, $x_5 = x_1 \cdot x_2$

这样上式即可化为以下五元一次线性回归方程：

$$\hat{y} = a + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5$$

根据推导结果来看，可以按多元线性回归方法计算各偏回归系数，建立二元二次多项式回归方程。

下面用 Python 求解二元二次多项式回归方程。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-20.py
#y=b1*x1+b2*(x2^2)+b3*x1*x2
import numpy as np

z=np.matrix([3,1.4,1.9]).T
myx =np.matrix([[7,3],[3,17],[11,5]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,1]**2,myx[0,0]*myx[0,1]],\
               [myx[1,0],myx[1,1]**2,myx[1,0]*myx[1,1]],\
               [myx[2,0],myx[2,1]**2,myx[2,0]*myx[2,1]]],\
               dtype=np.float64)
y =x*z
b=(x.T*x).I*x.T*y
print u"参数项矩阵为{0}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T

print "y=%f*x1+%f*x2^2+%f*x1*x2+%f"%(cb[0],cb[1],cb[2],mye)
```

程序计算回归方程的参数后，列出回归方程，如下所示：

```
参数项矩阵为[[ 3. ]
              [ 1.4]
              [ 1.9]]
y=3.000000*x1+1.400000*x2^2+1.900000*x1*x2+-0.000000
```

7.6.5 岭回归方法

只有正方形 ($n \times n$) 的矩阵（也称为方阵），才可能有但非必然有逆矩阵。若方阵 A 的

逆矩阵存在，则称 A 为非奇异方阵或可逆方阵。回顾一下参数项的求解公式：

$$\hat{B} = (X'X)^{-1}X'Y$$

这样就产生了一个问题：如果奇异矩阵或非方阵的矩阵不存在逆矩阵，即上式中的逆矩阵不存在了，怎样求解参数项呢？

此外，某些自变量之间可能存在共线性，这也给求逆带来了麻烦。岭回归方法可以解决这个问题，它是一种专用于共线性数据分析的有偏估计回归方法，实质上它是一种改良的最小二乘估计法，是通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价，获得回归系数更为符合实际、更可靠的回归方法，对病态数据的耐受性远远强于最小二乘法。其基本原理是：

给参数项求解公式中的 $X'X$ 部分加上正常数矩阵 kI ($k > 0$)，则 $X'X+kI$ 接近奇异的程度较小，如果 $X'X+kI$ 可逆，则参数的求解公式变为：

$$\hat{B}(k) = (X'X+kI)^{-1}X'Y$$

其中 k 称为岭参数。 $\hat{B}(k)$ 作为 B 的估计应比最小二乘估计 \hat{B} 稳定，当 $k=0$ 时，岭回归估计就是普通的最小二乘估计。因为岭参数 k 不是唯一确定的，所以得到的岭回归估计 $\hat{B}(k)$ 实际是对回归参数 B 的估计值。

选择一个适合的岭参数很重要，岭迹法是方法之一。选择的原则是：

- 各回归系数的岭估计基本稳定。
- 系数符号变得合理。
- 残差平方和。
- 取使方程基本稳定的最小 K 值。

7.6.6 伪逆方法

除了岭回归方法，伪逆方法也是一种不错的方法，它是对逆阵的推广。一般所说的伪逆是指 MoorePenrose 伪逆，它是由 E. H. Moore 和 Roger Penrose 分别独立提出的。

一个与 A 的转置矩阵 A^T 同型的矩阵 X ，满足： $AXA=A$ ， $XAX=X$ 。此时，称矩阵 X 为矩阵 A 的伪逆，也称为广义逆矩阵。

numpy 提供了求伪逆的函数 `linalg.pinv()`，可调用 numpy 的这个方法对一组数据进行回归分析，这组数据不适合用前面介绍的非线性回归算法计算。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-21.py
#y=b1*x1+b2*x2+b3*(x1^2)+b4*(x2^2)+b5*x1*x2
import numpy as np

z=np.matrix([1.4,1.9,1.7,0.8,1.1]).T
myx =np.matrix([[7,3],[3,17],[11,5]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,1],myx[0,0]**2,myx[0,1]**2,myx[0,0]*myx[0,1]],\
```

```

[myx[1,0],myx[1,1],myx[1,0]**2,myx[1,1]**2,myx[1,0]*myx[1,1]],\
[myx[2,0],myx[2,1],myx[2,0]**2,myx[2,1]**2,myx[2,0]*myx[2,1]],\
dtype=np.float64)

y =x*z
wn=np.linalg.pinv(x.T*x)
b=wn*x.T*y
print u"参数项矩阵为{0}".format(b)
i=0
cb=[]
while i<5:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T

print "y=%f*x1+%f*x2+%f*(x1^2)+%f*(x2^2)+%f*x1*x2+%f"%(cb[0],cb[1],cb[2],cb[3],c
b[4],mye)

```

程序运行后，输出的参数矩阵及回归方程如下：

参数项矩阵为[[1.59659657]

```

[ 0.69002152]
[ 2.01029166]
[ 0.9820693 ]
[ 0.4052783 ]]

```

```

y=1.596597*x1+0.690022*x2+2.010292*(x1^2)+0.982069*(x2^2)+0.405278*x1*x2+-
0.000000

```

7.7 PCA 降维

PCA 主要用于数据降维。由一系列特征组成的多维向量，其中某些元素本身没有区分性，比如某个元素在所有的样本中都相等，或者彼此差距不大，那么这个元素本身就没有区分性，如果用它做特征来区分，贡献会非常小。我们的目的是找到那些变化大的元素，即方差大的维，而去除掉那些变化不大的维。

使用 PCA 的好处在于，可以对新求出的“主元”向量的重要性进行排序。根据需要取前面最重要的部分，将后面的维数省去，从而达到降维、简化模型或对数据进行压缩的效果。同时最大程度地保持了原有数据的信息，较低的维数意味着运算量的减少，在数据较多的情况带来的性能提高更明显。

PCA 通过将主成分分析的问题转化为求解协方差矩阵的特征值和特征向量来计算。其目标是寻找 $r(r < n)$ 个新变量，使它们反映事物的主要特征，压缩原有数据矩阵的规模，每个新变量是原有变量的线性组合，体现原有变量的综合效果，这 r 个新变量称为“主成分”，它们可以在很大程度上反映原来 n 个变量的影响，并且这些新变量是互不相关的，也是正交的。

以将数据的维数从 N 降到 5 为例，PCA 算法过程如下：

(1) 计算样本矩阵 X 协方差矩阵。

(2) 计算协方差矩阵 S 的特征向量 e_1, e_2, \dots, e_N 及其特征值 $i = 1, 2, \dots, N$ 。

(3) 把特征值按从大到小排序，取前 5 位特征值对应的特征向量组成投影矩阵 W 。

(4) 投影数据到 W 组成的空间之中。

上述算法过程较抽象，理解它的最好方式就是动手实现它。下面编写一段 Python 程序，使用 mlpv 库提供的 PCA 类，实现 PCA 算法。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#7-22.py
import numpy as np
import matplotlib.pyplot as plt
import mlpv
np.random.seed(0)
mean, cov, n = [0, 0], [[1, 1], [1, 1.5]], 100
x = np.random.multivariate_normal(mean, cov, n)
pca = mlpv.PCA()
pca.learn(x)
coeff = pca.coeff()
fig = plt.figure(1)
plot1 = plt.plot(x[:, 0], x[:, 1], 'o')
plot2 = plt.plot([0, coeff[0, 0]], [0, coeff[1, 0]], linewidth=4, color='r')
plot3 = plt.plot([0, coeff[0, 1]], [0, coeff[1, 1]], linewidth=4, color='g')
xx = plt.xlim(-4, 4)
yy = plt.ylim(-4, 4)
z = pca.transform(x, k=1)
xnew = pca.transform_inv(z)
fig2 = plt.figure(2)
plot1 = plt.plot(xnew[:, 0], xnew[:, 1], 'o')
xx = plt.xlim(-4, 4)
yy = plt.ylim(-4, 4)
plt.show()
```

如图 7-35 所示是程序生成的两张散点图，左边是数据降维前的效果，右边是数据降维后的效果。能看出数据降维后，数据的整体趋势并没有改变，最大程度地保持了原有数据的信息。

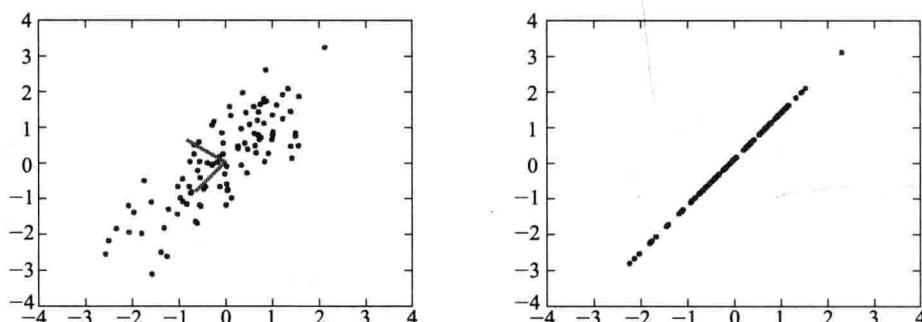


图 7-35 数据降维

7.8 小结

机器学习算法是机器学习的灵魂。本章首先介绍神经网络，由浅入深地讲述了基于 Rosenblatt 感知器的线性神经网络、反向传播算法及基于多层感知器的非线性神经网络；然后介绍了平均值与方差、贝叶斯分类等统计算法；接着阐述了相似度算法的两个常用方法：欧氏距离和余弦相似度；最后讲解了 SVM、回归算法、PCA 降维等机器学习算法。

在讲述算法的同时，笔者理论联系实际，用 Python 实现了本章涉及的大部分机器学习算法。此外，为了帮助读者更好地理解，在解说每一节的算法之前，介绍了算法涉及的数学基础知识。

每个机器学习算法不一定适合所有的机器学习任务，因此，应在实践中应用这些算法，观察算法的实际效果，选择最适合的算法。

思考题

(1) 编写 Python 代码实现一个 Rosenblatt 感知器，随机产生一组 X 和 Y 的样本值，这些样本值分别属于以下两类函数：

$$5x - 3 = y \text{ 为第 1 类}$$

$$2x + 6 = y \text{ 为第 2 类}$$

然后用随机生成的测试数据进行分类，并绘制出分类线。

(2) 编写 Python 代码实现多层感知器，随机产生一组 X 和 Y 的样本值，用随机数据对训练后的网络进行测试绘制散点图和误差曲线，计算 X 和 Y 的方差和平均值，分析其分布趋势。

(3) 编写 Python 代码实现多层感知器，绘制样本散点图和误差曲线。随机产生一组 X 样本值，并将 Y 的样本值函数定义为：

$$Y = \sin(x) * 0.7$$

(4) 编写 Python 代码实现 SVM 算法，随机产生一组 X 和 Y 的样本值，这些样本值分别属于以下两类函数：

$$y = x^a + b \quad (a \leq 3, \text{abs}(b) < 20) \text{ 为第一类}$$

$$y = x^a + b \quad (a \geq 5, \text{abs}(b) < 10) \text{ 为第二类}$$

然后用随机生成的测试数据进行分类，并绘制出散点图。

(5) 选择一组数据，用 Python 实现下面的回归方程：

$$y = b_1 * x_2 + b_2 * x_1^2 + b_3 * x_1 * x_2$$

第8章

数据拟合案例

在科学和工程问题上可通过采样、实验等方法获得若干离散的数据，分析这些数据能得到一个连续的函数（曲线）或者更加密集的离散方程，且这个方程与已知数据相吻合，这个过程叫做数据拟合。数据实质是针对一组未知规律的数据建立数学方程，通过数学方法建立一个函数映射方式。

前面几章涉及了很多回归分析算法，都是先根据样本建立模型，然后分析回归效果，最终目的是弄清楚两个或两个以上变量之间的因果关系，因此，这些算法都是对数据的拟合。

8.1 数据拟合

分析一个自变量和一个因变量之间的关系，可通过图像分析法与神经网络拟合法建立数学模型进行映射。

8.1.1 图像分析法

顾名思义，图像分析法就是将样本数据中的自变量 X 和因变量 Y 的值映射为平面直角坐标系中的点，其 X 轴坐标和 Y 轴坐标的值分别为自变量和因变量的值，这些点共同形成一个散点图。通过分析散点图中这些点的几何分布趋势，对照常见函数图像，选择最接近的、最适合的数学函数作为拟合方程。

既然是图像分析法，那么就需要对常见的数学函数及其图像有一个直观的认识。因此，这里先介绍常见的数学函数。

1. 幂函数

幂函数是形如 $f(x)=x^a$ 的函数，函数以底数为自变量，幂为因变量，指数 a 为常量， a 可以是自然数、有理数、任意实数或复数。下面是几个经典的幂函数。

(1) 指数为 1、2、1/2 的幂函数，它们分别为 $y=x$ 、 $y=x^2$ 、 $y=x^{1/2}$ ，图像如图 8-1 所示。

(2) 指数为 3、1/3 的幂函数，它们分别为 $y=x^3$ 、 $y=x^{1/3}$ ，图像如图 8-2 所示。

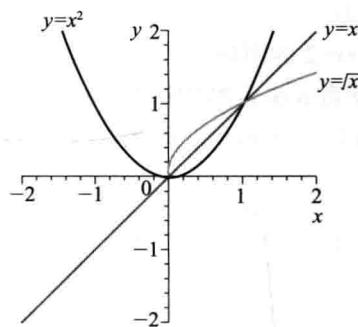


图 8-1 指数为 1、2、1/2 的幂函数图像

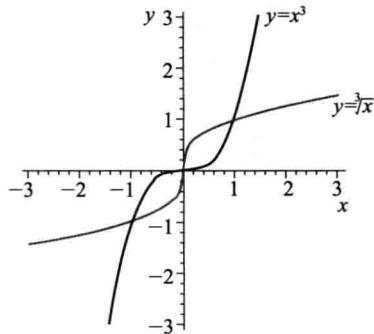


图 8-2 指数为 3、1/3 的幂函数图像

(3) 指数为 -1 的幂函数，函数为 $y=x^{-1}$ ，图像如图 8-3 所示。

2. 一次函数

一次函数又称线性函数，是指拥有一个变量的一阶多项式函数。一次函数可以表达为以下斜截式的格式：

$$f(x)=kx+b$$

其中， k 是斜率， b 是截距。

一次函数在二维坐标系中表现为一条直线，以自变量为 X 轴，以因变量为 Y 轴。如： $y=2x+1$ 和 $y=-x+1$ 属于一次函数，它们的图像如图 8-4 所示。

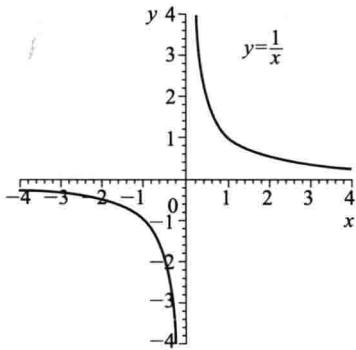


图 8-3 指数为 -1 的幂函数图像

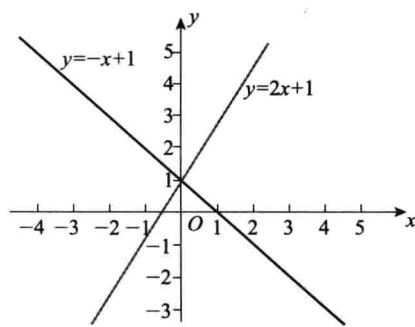


图 8-4 一次函数图像

3. 一元二次函数

如果 $y=ax^2+bx+c$ (a 、 b 、 c 为常数， a 不为 0)，则称 y 为 x 的一元二次函数，其中， a 称为二次项系数， b 为一次项系数， c 为常数项。

一元二次函数是抛物线，是轴对称图形，它的对称轴为直线 $x=-\frac{b}{2a}$ ，它的顶点坐标为 $\left(-\frac{b}{2a}, \frac{4ac-b^2}{4a}\right)$ 。此外，一元二次函数还可写成交点式： $y=a(x-x_1)(x-x_2)$ ，交点式仅限于与 x

轴有交点的抛物线，与 x 轴的交点坐标是 $(x_1, 0)$ 和 $(x_2, 0)$ 。

如图 8-5 所示为一元二次函数 $y=2x^2+x+1$ 和 $y=-2x^2+x+2$ 的图像。从图像上观察，一元二次函数是一个开口向上或开口向下的抛物线，二次项系数 a 决定抛物线的开口方向和大小，当 $a>0$ 时，抛物线向上开口；当 $a<0$ 时，抛物线向下开口。 $|a|$ 越大，则抛物线的开口越小。

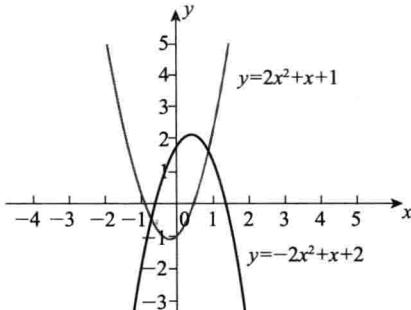


图 8-5 二次函数图像

4. 指数函数

指数函数 $y=e^x$ 是数学中重要的函数，这里的 e 是数学常数，就是自然对数的底数，近似等于 2.718281828。 $y=e^x$ 的图像总在 x 轴之上并从左向右递增，它接近 x 轴但不会与 x 轴相交。

此外，还可定义更一般的指数函数 $y=a^x$ ，对于 $a>0$ 和实数 x ，该函数可称为底数为 a 的指数函数。 a^x 可如下变换为以 e 为底的指数函数。

$$a^x = (e^{\ln a})^x = e^{x \ln a}$$

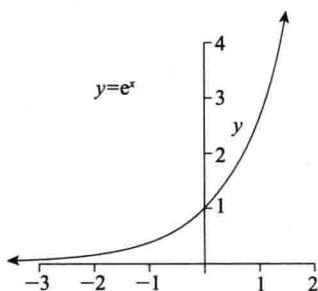


图 8-6 指数函数 $y=e^x$

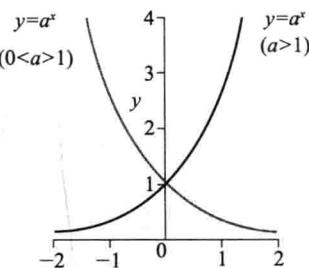


图 8-7 指数函数 $y=a^x$

观察图 8-7 所示的指数函数 $y=a^x$ ，当 $a>1$ 时，从左到右函数是递增的，否则函数是递减的。指数函数遵循以下运算规律：

$$\begin{aligned} a^0 &= 1 \\ a^1 &= a \\ a^{x+y} &= a^x a^y \end{aligned}$$

$$a^{xy} = (a^x)^y$$

$$\frac{1}{a^x} = \left(\frac{1}{a}\right)^x = a^{-x}$$

$$a^x b^x = (ab)^x$$

5. 对数函数

相对于底数 α 数 x 的对数是 α^y 的指数 y , 使得 $x=\alpha^y$, 相对于底数 α 的数 x 的对数通常记为 $y=\log_a x$, 在工程计算中常用 e 、10 和 2 来做底数, 如表 8-1 所示。

表 8-1 常用对数函数表示

底数	名称	表示
2	二进制对数	$\lg x$
10	常用对数	$\lg x$
e	自然对数	$\ln x$

观察图 8-8 所示的对数函数图像, 可看出: 当 $\alpha>1$ 时, 函数是递增的, 否则, 函数是递减的。自然对数的底 e 大于 1, 它的函数图像是递增的, 如图 8-9 所示。

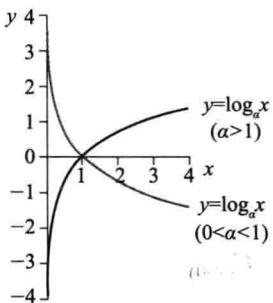


图 8-8 $y=\log_a x$

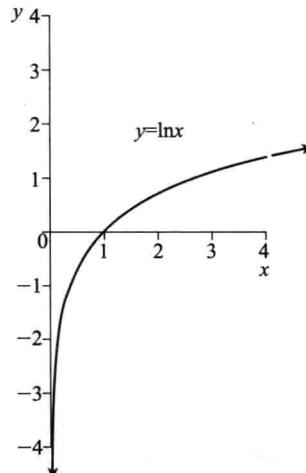


图 8-9 自然对数

6. 三角函数

三角函数中使用得最多的是正弦 (如图 8-10 所示)、余弦 (如图 8-10 所示)、正切 (如图 8-11 所示)、余切 (如图 8-12 所示)。

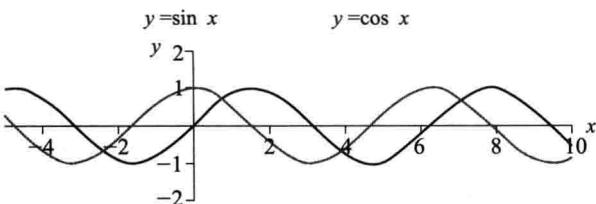


图 8-10 正弦函数与余弦函数

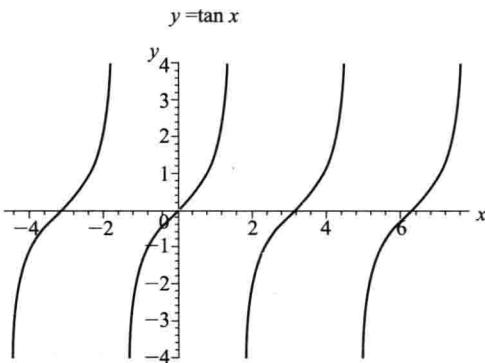


图 8-11 正切函数

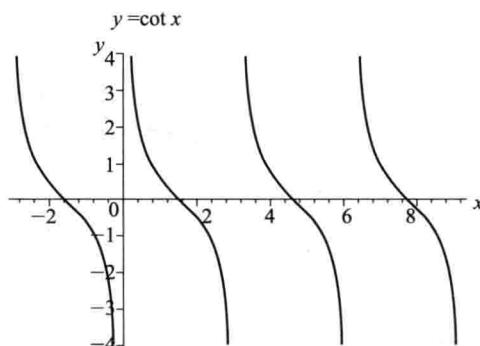
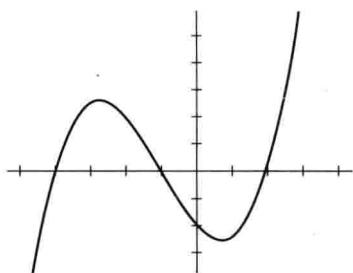
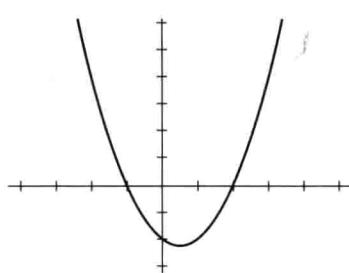


图 8-12 余切函数

7. 多项式函数

多项式的函数图像呈现连续曲线，以多项式 $y = \frac{x^3}{4} + \frac{3x^3}{4} - \frac{3x}{2} - 2$ 和 $y = x^2 - x - 2$ 为例，其函数图像如图 8-13 和图 8-14 所示。

图 8-13 $y = \frac{x^3}{4} + \frac{3x^3}{4} - \frac{3x}{2} - 2$ 的函数图像图 8-14 $y = x^2 - x - 2$ 的函数图像

仔细观察前面列举的函数图像，可发现常见的函数分为两种：线性函数与非线性函数。其中，线性函数是变量与自变量成一次方的函数关系，在函数图上呈现一条直线；而非线性函数是指两个变量间的关系不成简单比例，函数图像呈现为曲线。

线性拟合是以一条直线来拟合自变量与因变量的关系，而非线性拟合用连续曲线来拟合自变量与因变量之间的关系。下面以几个具体实例说明图像分析法在线性与非线性拟合中的应用。

(1) 多项式是非线性函数，如图 8-15 所示的图像为两类样本数据生成的散点图，其中虚线和实线各代表一类。

从图 8-15 看， X 轴的区域为 $[0, 5]$ ，两条曲线代表的数据都比较接近多项式函数图像 $y = x^a + b$ ，因此，可选用幂函数作为数据拟合的模型。此外，从图像可判定，虚线函数的导数要比实线函数的导数增长得快，虚线数据表示的方程参数 a 要比实线表示的方程参数 a 大，原因是：导数即切线的斜率，导数越大，切线越陡，而虚线函数的切线明显要比实线函数陡很多，它的曲线在后期更加上扬。事实上，图 8-15 所代表的两类数据的模型分别为：虚线

是 $y=x^5+6$, 实线是 $y=x^3+6$ 。

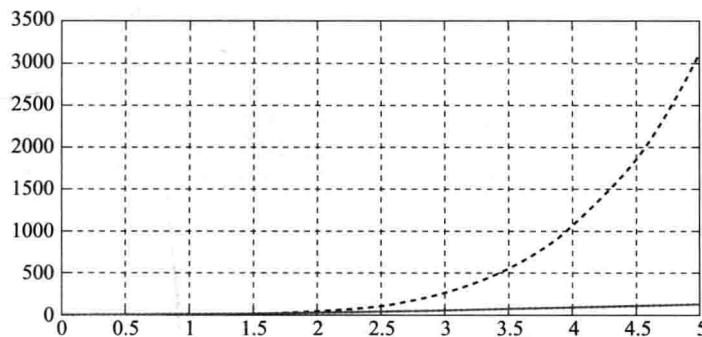


图 8-15 多项式非线性拟合

(2) 大学生身高与体重线性拟合。从某大学中随机选取 9 名女大学生, 其身高和体重数据如表 8-2 所示。下面分析这些数据并建立女大学生身高与体重模型, 从而指导女大学生通过简易的公式查询自己的体重是否标准。

表 8-2 女大学生身高与体重数据

编 号	1	2	3	4	5	6	7	8	9
身高 /cm	160	165	158	172	159	176	160	162	171
体重 /kg	58	63	57	65	62	66	58	59	62

首先, 使用 R 语言输入数据 (本例只有 9 个数据, 可手工录入。当数据量很大时, 可使用第 6 章介绍的 `read.table` 函数读取数据)。

```
>c(58,63,57,65,62,66,58,59,62)->y  
>c(160,165,158,172,159,176,160,162,171)  
->x
```

接着, 用自变量身高 (x) 与因变量体重 (y) 组成 9 个数据点, 绘制在直角坐标系中, 生成散点图。

观察分析散点图趋势, 图 8-16 表明, 随着 x 的增长, y 也增长, 数据点呈一条直线分布 (线性拟合)。

下面以直线模型对数据进行拟合, 可通过 R 语言进行拟合分析。

```
> lm(y~x)->xy  
> summary(xy)  
Call:  
lm(formula = y ~ x)
```

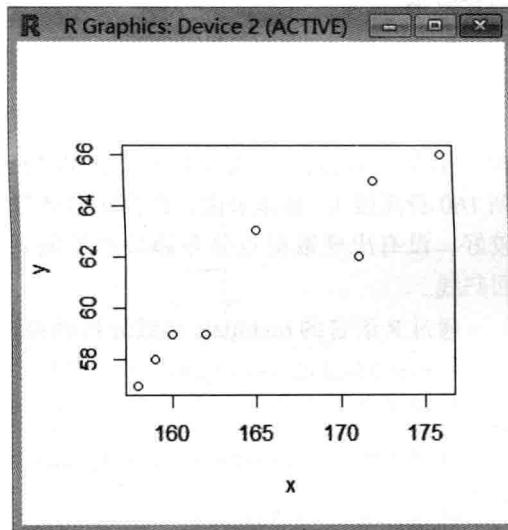


图 8-16 女大学生身高体重散点图

Residuals:

	Min	1Q	Median	3Q	Max
	-1.7317	-1.0989	-0.9412	0.8471	3.3223

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-8.28830	15.94636	-0.520	0.61926
x	0.42117	0.09671	4.355	0.00333 **

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.808 on 7 degrees of freedom

Multiple R-squared: 0.7304, Adjusted R-squared: 0.6919

F-statistic: 18.97 on 1 and 7 DF, p-value: 0.003334

通过对上述直线方程模型的分析，可初步得出以下结论：

- 直线方程即线性方程，可写成 $y=ax+b$ 的方式， a 即系数（Coefficients 项中的 x ）为 0.42117， b 即截距（Coefficients 项中的 Intercept）为 -8.28830，线性拟合模型为 $y=0.42117x-8.28830$ 。
- Coefficients 栏的 x 行尾的星号为 2 个。星号的含义表示线性关系是否显著：* 的数量是 0~3，* 的数量越多则线性关系越显著。在本例中，自变量身高与因变量体重的线性关系并不是非常显著，否则星号应为 3 个。

下面绘制回归线，图 8-17 所示。观察各数据点在回归线周围的分布情况，目测拟合效果。

```
> plot(x, y)
> abline(lm(y~x))
```

从图 8-17 来看，数据分布在回归线周围，但某些数据点的残差较大（比如 155 到 160 身高段）。总体来说，数据拟合效果较好，没有出现数据点分布趋势严重偏离回归线。

通过 R 语言的 residuals 函数分析残差。

```
> residuals(xy) -> xy_res
> xy_res
 1           2           3           4           5           6           7
-1.0988557  1.7952956 -1.2565162  0.8471074  3.3223140  0.1624285 -1.0988557
 8           9
-0.9411952 -1.7317228
```

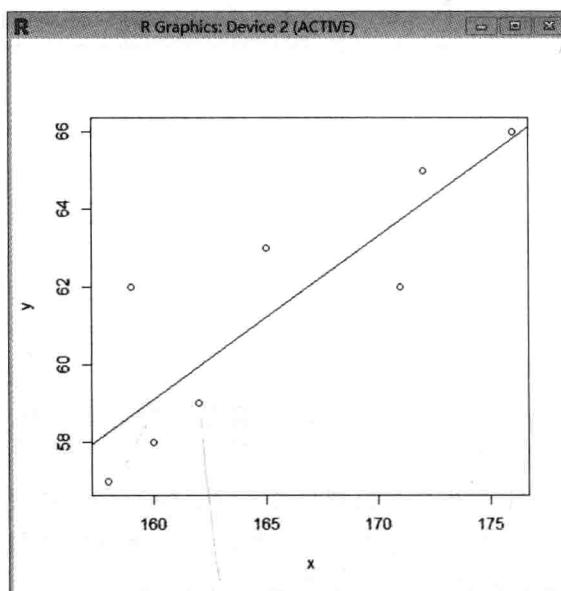


图 8-17 女大学生身高体重数据的回归线

残差计算的方式为：通过上一步计算得到的线性模型，将 9 个女大学生的自变量身高带

入线性模型中的 x ，计算出观测的 y 值后，计算实际值与预测 y 值的差额，这个差额就是残差。如果拟合模型正确，可以将残差看作误差的观测值，它应符合模型的假设条件，且具有误差的一些性质。

上述代码根据线性模型计算得出残差对象 `xy_res`，9 个元素分别为 9 个女大学生样本数据的残差，其中残差最大的数据为第 5 个女大学生，即 3.3223140，最小为 0.1624285，是第 6 个女大学生。

然后，计算残差平方和。残差平方和是指每个残差的平方后的累计，它表示随机误差的效应。残差平方和是衡量拟合优度的重要标准，它的值越小，说明拟合效果越好。通过下面代码计算，线性模型拟合后的残差平方和为 22.88334。

```
> sum(xy_res*xy_res)
[1] 22.88334
```

最后，分析残差分布、标准残差及标准残差图。

残差为测定值与按回归方程预测的值之差，可用 δ 表示，如果数据拟合模型效果较好，那么残差 δ 应遵从正态分布 $N(0, \sigma^2)$ ，通过绘制 QQ 图观察残差数据是否符合正态分布。

```
> qqnorm(xy_res)
> qqline(xy_res)
```

观察残差分布 QQ 图，如图 8-18 所示。可看出：残差 δ 分布接近正态分布，数据点基本在直线附近，回归效果较理想，但仍不是非常理想。

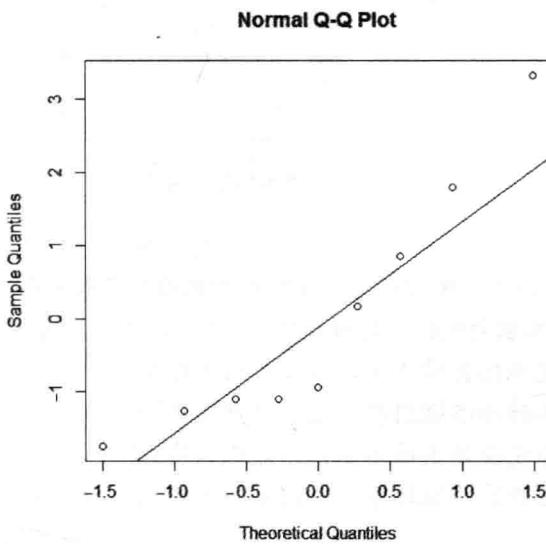


图 8-18 残差分布 QQ 图

标准残差就是（残差 δ - 残差的均值）/ 残差的标准差，用 δ^* 表示。下面通过 R 语言的 `rstandard` 函数计算身高体重拟合模型的标准残差，同时绘制标准残差图，图 8-19 所示。

```

> rstandard(xy) -> std_xy_res
> std_xy_res
      1          2          3          4          5          6          7
-0.6696927  1.0532610 -0.7984996  0.5447647  2.0629420  0.1235619 -0.6696927
      8          9
-0.5591209 -1.0857787
> plot(x, std_xy_res)
> abline(h=0)

```

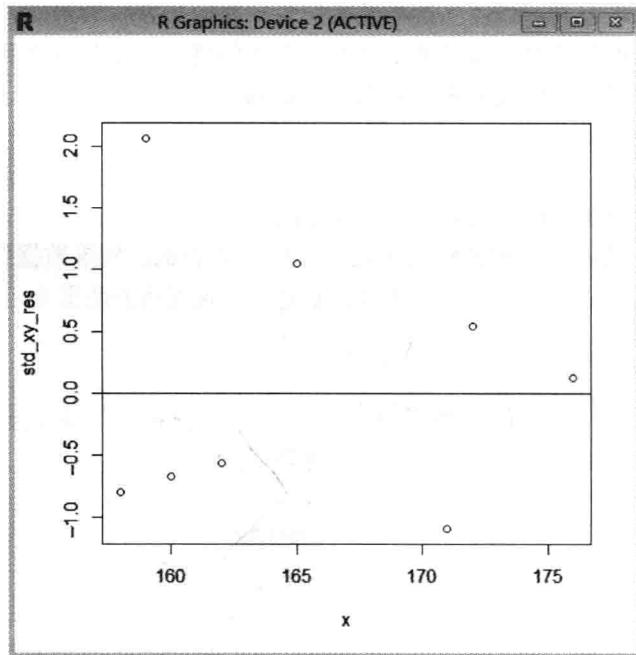


图 8-19 身高体重标准残差图

标准残差图是以拟合模型的自变量为横坐标，以标准残差为纵坐标，将每一个自变量的标准残差描述在该平面坐标上，形成图形，实验点的标准残差落在残差图的 $(-2, 2)$ 区间以外的概率 ≤ 0.05 ，若某一实验点的标准化残差落在 $(-2, 2)$ 区间以外，可在95%置信度将其判为异常实验点。置信度也称置信水平，抽样不可能抽取全部总体进行分析，由于样本的随机性，通过这些样本对总体参数作出估计时，结论总是不确定的，因此引入置信度，用概率来陈述总体参数值落在样本统计值某一区内的可能性。

当描绘标准残差的点围绕标准残差等于0的直线上下完全随机地分布，绝大多数点落在 $(-2, +2)$ 的水平带状区间之间，且不带有任何系统趋势时，则说明拟合模型对原观测值（即样本）的拟合情况良好，如图8-20所示。

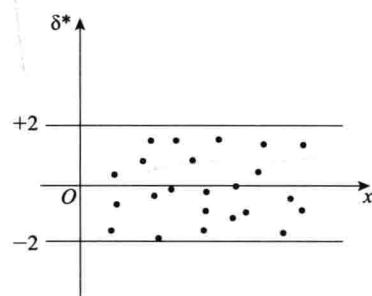


图 8-20 拟合良好的标准残差图

如果拟合方程原本是非线性模型（曲线），但拟合时却采用了线性模型（直线），标准化残差图就会表现出曲线形状，产生系统性偏差，说明拟合模型不适合，如图 8-21、图 8-22 所示。

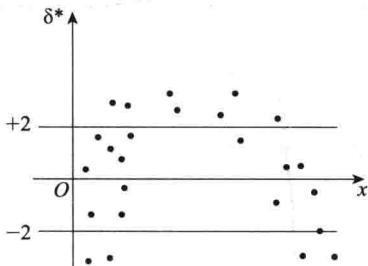


图 8-21 出现系统性偏差的标准残差图①

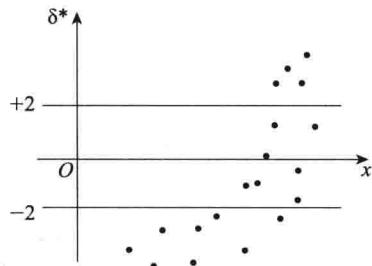


图 8-22 出现系统性偏差的标准残差图②

针对当前的拟合模型，样本数据中如果出现了异常点，它们会远离大多数数据点，如图 8-23 所示。

此外，如果拟合不充分，很多点会落在 $(-2, +2)$ 的水平带状区间之外，如图 8-24 所示。

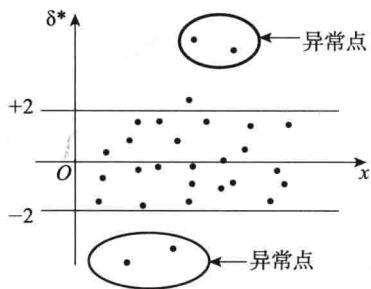


图 8-23 异常点

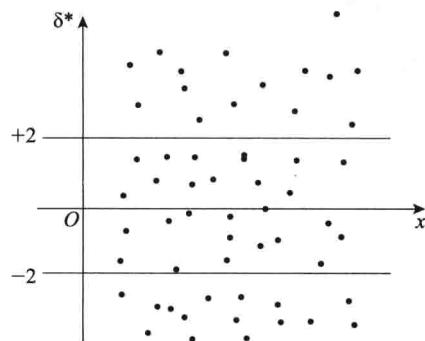


图 8-24 拟合不充分

根据标准残差图的相关知识，观察图 8-19 所示的身高体重标准残差图，可得出以下结论：

- 绝大部分数据在 $(-2, +2)$ 的水平带状区间内，因此模型拟合较充分。
- 数据点分布稍均匀，但没有达到随机均匀分布的状态。此外，部分数据点还是呈现某种曲线波动形状，有少许系统性偏差，因此可能采用非线性拟合效果会更好。
- 大学生身高与体重非线性拟合。继续以女大学生身高与体重的例子为例，再次仔细观察图 8-16 所示的散点图，数据点呈现上凸的递增曲线分布，可对身高与体重的关系应用非线性模型，也许效果会比直线好，因为曲线更贴近它的分布趋势。那么选用哪个非线性数学函数比较适合呢？观察前面列举的常见函数图像，可发现幂函数比较适合，尤其是 $y=x^{1/2}$ 的函数图像，也呈现上凸且递增趋势，但是幂函数 $y=x^a$ 仅有一个参数 a ，需要进行变形，在函数尾部加上截距 b ，使函数图像能在 Y 轴整体上下移动，这样，就拥有了两个可调节参数，形成的拟合方程更灵活和适用。

细心的读者肯定发现了，加上截距后，幂函数 $y=x^a$ 变成了多项式函数 $y=x^a+b$ ，因此将非线性拟合模型假设为 $y=x^a+b$ 的多项式方程。

首先，显示自变量和因变量数据。

```
> x
[1] 160 165 158 172 159 176 160 162 171
> y
[1] 58 63 57 65 62 66 58 59 62
```

接着，通过 nls 函数建立非线性拟合模型，并通过 summary 函数分析该模型。

```
> nls(y ~ Const + x^A) -> nlmod
> summary(nlmod)

Formula: y ~ Const + x^A

Parameters:
Estimate Std. Error t value Pr(>|t|)
Const -19.66594 15.09467 -1.303 0.234
A 0.86036 0.03657 23.523 6.37e-08 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.808 on 7 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 5.78e-06
```

然后，计算残差，分析残差平方和。

```
> residuals(nlmod) -> nlxy_res
> nlxy_res
[1] -1.0986204 1.7882691 -1.2508068 0.8448399 3.3251002 0.1704208 -1.0986204
-0.9449555 -1.7357098
attr(,"label")
[1] "Residuals"
> sum(nlxy_res * nlxy_res)
[1] 22.88108
```

上述代码计算得出的残差平方和为 22.88108，比刚才应用线性拟合的残差平方和 22.88334 稍小些，可见非线性模型更适合拟合本例中的身高体重数据。

下面，绘制残差的 QQ 图，如图 8-25 所示。从 QQ 图中，可看出：残差 δ 分布接近正态分布，数据点基本在直线附近。

```
> qqnorm(nlxy_res)
> qqline(nlxy_res)
```

接着，计算标准残差，将计算结果存入 std_nlxy_res。

```
> (nlxy_res - mean(nlxy_res)) / sd(nlxy_res) -> std_nlxy_res
> std_nlxy_res
[1] -0.6496072 1.0574063 -0.7395947 0.4995580 1.9661322 0.1007751
[7] -0.6496072 -0.5587453 -1.0263171
```

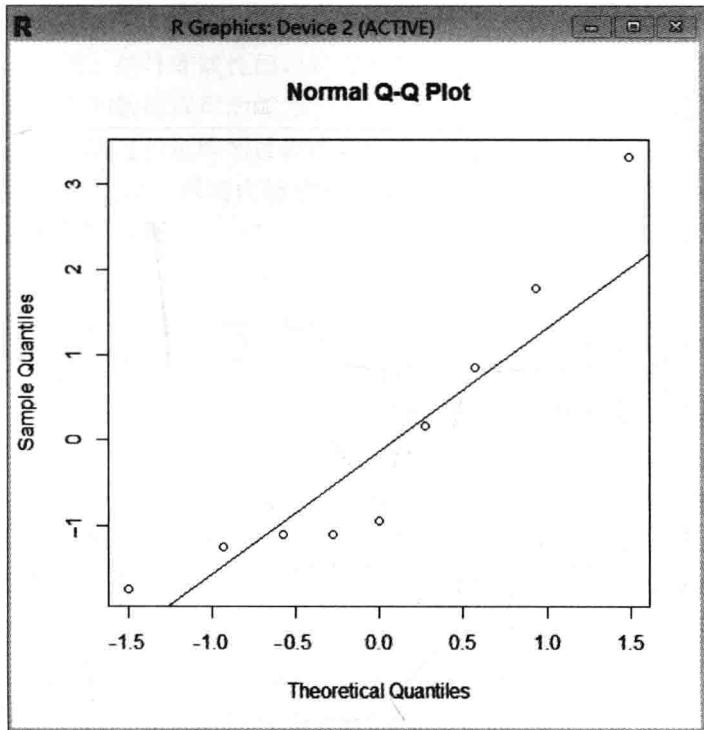


图 8-25 残差的 QQ 图

```
attr(),"label")
[1] "Residuals"
```

绘制标准残差图，如图 8-26 所示。

```
> plot(x, std_nlxy_res)
> abline(h=0)
```

通过绘制图 8-26 所示的标准残差图，应用 summary 函数对非线性模型进行分析，以及计算残差平方和与标准残差，可得出以下结论：

- 图 8-26 中没有任何异常点，计算的标准残差全在 (-2, +2) 的水平带状区间内，模型拟合充分。
- 图 8-26 中数据点分布较均匀，拟合效果良好。
- 残差平方和比线性模型小，非线性模型更适于描述女大学生身高与体重的关系。
- 非线性拟合模型为 $y=x^{0.86036}-19.66594$ 。

最后，观察一下拟合预测数据以及拟合效果图，如图 8-27 所示。

```
> predict(nlmod)
[1] 59.09862 61.21173 58.25081 64.15516 58.67490 65.82958 59.09862 59.94496
[9] 63.73571
> plot(x,y)
> lines(x, predict(nlmod), col = 2)
```

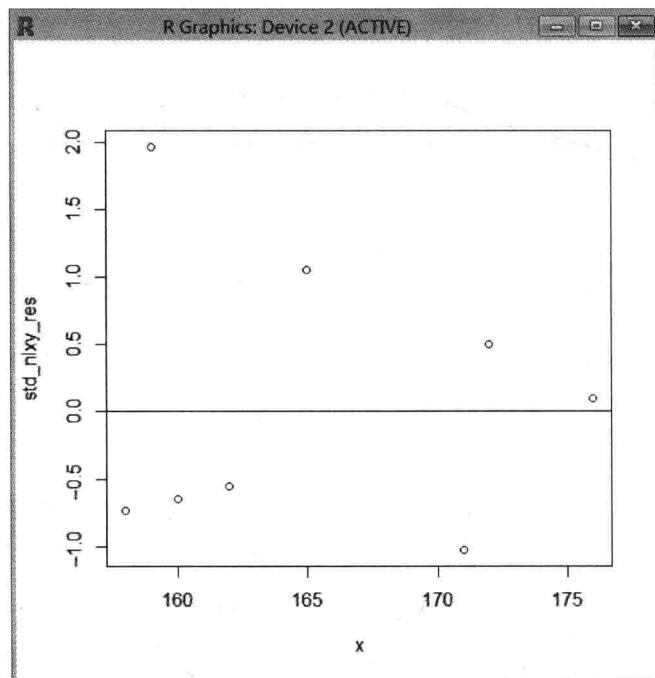


图 8-26 身高体重非线性模型标准残差图

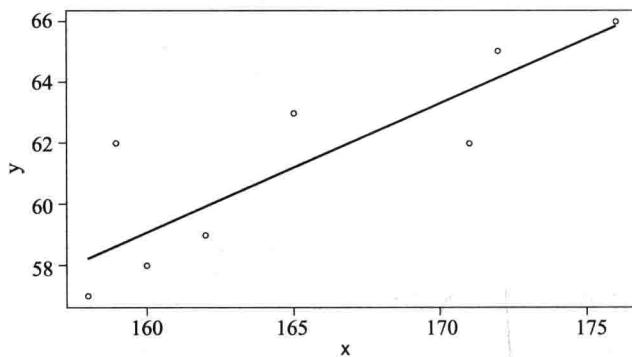


图 8-27 拟合效果图

不要把图 8-27 中的拟合线误看成直线，通过分析 `predict(nlmod)` 的执行结果及得到的非线性拟合方程可看出，拟合预测点的位置在一条曲线上，但位置相差不大。

仔细观察图 8-27（非线性拟合）与图 8-17（线性拟合）的拟合线，可以看出，图 8-27 所示的确实是一条曲线。

8.1.2 神经网络拟合法

数据分布如果近似一条直线，可以使用线性神经网络来完成数据拟合，比如上一章介绍

的 Rosenblatt 感知器；如果呈曲线，可使用多层感知器的神经网络进行拟合。上一节讲述了通过观察数据分布图像，估计非线性回归对应的数学方程。本节将要介绍的神经网络方法与之不同，它具有学习未知数据规律的能力。

通过样本对神经网络进行训练的过程就是在输入与输出数据之间建立映射的过程。训练完成后，神经网络就已定型，网络内部的权值矩阵和神经元的激活函数共同组成了映射组件，这些组件代替了数学方程。

1. 常见数学函数拟合

理论上来说，有了神经元作为映射组件，神经网络可以对数据之间的任何规律进行学习和模仿。下面是神经网络对上一节介绍的几个数学函数进行拟合的效果。

(1) sin 函数拟合。顾名思义，sin 函数拟合就是使用一组数据 x 作为输入，通过神经网络建立模型，其输出值为 $\sin(x)$ 。可通过使用 Python 编写神经网络，实现 sin 函数拟合。

在华章网站下载本书的资源包，打开里面的文档“多层感知器神经网络源代码.doc”。下面将在该文档中源代码的基础上进行修改，实现非线性拟合。

首先，随机生成 500 个 x 值，同时计算这些样本对应的目标值。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-1.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy

isdebug=False
#x和d样本初始化
train_x = []
d = []
for yb_i in xrange(0,500):
    train_x.append([np.random.rand()*4*np.pi-2*np.pi])
for yb_i in xrange(0,500):
    d.append(np.sin(train_x[yb_i]))
```

然后设置相关参数。代码如下：

```
warray_tnx=len(train_x[0])
warray_n=warray_tnx*4*2
```

```
#基本参数初始化
oldmse=10**100
err=[]
maxtrycount=800

mycount=0.0
if maxtrycount>=20:
    r=maxtrycount/10
```

```

else:
r=maxtrycount/2
#sigmoid函数
ann_sigfun=None
ann_delta_sigfun=None
#总层数初始化
alllevel_count=int(warray_txn*4*1.5+1)
# 非线性层数初始化
hidelevel_count=alllevel_count-1

#学习率参数
learn_r0=0.002
learn_r=learn_r0 *1.5
#动量参数
train_a0=learn_r0*1.2
train_a0*=0.001
train_a=train_a0
expect_e=0.02

```

接着，对数据进行预处理，并生成初始权值矩阵。

```

#对输入数据进行预处理
ann_max=[]
for m_ani in xrange(0,warray_txn):
    temp_x=np.array(train_x)
    ann_max.append(np.max(temp_x[:,m_ani]))
ann_max=np.array(ann_max)

def getnowsx(mysx,in_w):
    '''生成本次的扩维输入数据'''
    global warray_n
    mysx=np.array(mysx)
    x_end=[]
    for i in xrange(0,warray_n):
        x_end.append(np.dot(mysx,in_w[:,i]))
    return x_end

def get_inlw(my_train_max,w_count,myin_x):
    '''计算对输入数据预处理的权值'''
    #对随机生成的多个权值进行优化选择，选择最优的权值
    global warray_txn
    global warray_n
    mylw=[]
    y_in=[]
    #生成测试权值
    mylw=np.random.rand(w_count,warray_txn,warray_n)
    for ii in xrange(0,warray_txn):
        mylw[:,ii,:]=mylw[:,ii,:]*1/float(my_train_max[ii])-1/float(my_train_max[ii])*0.5

    #计算输出

```

```

for i in xrange(0,w_count):
    y_in.append([])
    for xj in xrange(0,len(myin_x)):
        y_in[i].append(getnewsx(myin_x[xj],mylw[i]))
#计算均方差
mymin=10**5
mychoice=0
for i in xrange(0,w_count):
    myvar=np.var(y_in[i])
    if abs(myvar-1)<mymin:
        mymin=abs(myvar-1)
        mychoice=i
#返回数据整理的权值矩阵
return mylw[mychoice]

mylnww=get_inlw(ann_max,300,train_x)

def get_inputx(mytrain_x,myin_w):
    '''将训练数据通过输入权数，扩维后形成输入数据'''
    end_trainx=[]
    for i in xrange(0,len(mytrain_x)):
        end_trainx.append(getnewsx(mytrain_x[i],myin_w))
    return end_trainx

x=get_inputx(train_x,mylnww)

def get_siminx(sim_x):
    global mylnww
    myxx=np.array(sim_x)
    return get_inputx(myxx,mylnww)

def getlevelw(myin_x,wo_n,wi_n,w_count):
    '''计算一层的初始化权值矩阵'''
    mylw=[]
    y_in=[]
    #生成测试权值
    mylw=np.random.rand(w_count,wi_n,wo_n)
    mylw=mylw*2.-1

    #计算输出
    for i in xrange(0,w_count):
        y_in.append([])
        for xj in xrange(0,len(myin_x)):
            x_end=[]
            for myii in xrange(0,wo_n):
                x_end.append(np.dot(myin_x[xj],mylw[i,:,:myii])))
            y_in[i].append(x_end)
    #计算均方差
    mymin=10**3
    mychoice=0
    for i in xrange(0,w_count):

```

```

        myvar=np.var(y_in[i])
        if abs(myvar-1)<mymin:
            mymin=abs(myvar-1)
            mychoice=i
    #返回数据整理的权值矩阵
    csmylw=mylw[mychoice]
    return csmylw,y_in[mychoice]

ann_w=[]
def init_annw():
    global x
    global hidelevel_count
    global warray_n
    global d
    global ann_w
    ann_w=[]

    lwyii=np.array(x)
    for myn in xrange(0,hidelevel_count):
        #层数
        ann_w.append([])
        if myn==hidelevel_count-1:
            for iii in xrange(0,warray_n):
                ann_w[myn].append([])
                for jjj  in xrange(0,warray_n):
                    ann_w[myn][iii].append(0.0)
        elif myn==hidelevel_count-2:
            templw,lwyii=getlevelw(lwyii,len(d[0]),warray_n,10)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,len(d[0])):
                    ann_w[myn][xii].append(templw[xii,xjj])
                for xjj in xrange(len(d[0]),warray_n):
                    ann_w[myn][xii].append(0.0)
        else:
            templw,lwyii=getlevelw(lwyii,warray_n,warray_n,10)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,warray_n):
                    ann_w[myn][xii].append(templw[xii,xjj])
    ann_w=np.array(ann_w)

def generate_lw(trycount):
    global ann_w
    print u"产生权值初始矩阵",
    meanmin=1
    myann_w=ann_w
    alltry=30
    tryc=0
    while tryc<alltry:
        for i_i in range(trycount):

```

```

print ".",
init_annw()
if abs(np.mean(np.array(ann_w)))<meanmin:
    meanmin=abs(np.mean(np.array(ann_w)))
    myann_w=ann_w
tryc+=1
if abs(np.mean(np.array(myann_w)))<0.01:break

ann_w=myann_w
print
print u"权值矩阵平均:%f"%(np.mean(np.array(ann_w)))
print u"权值矩阵方差:%f"%(np.var(np.array(ann_w)))

generate_lw(15)

```

下面，对所有样本数据进行反复训练，直到误差降到期望值为止。单个样本的训练过程如下：

```

def sample_train(myx,myd,n,sigmoid_func,delta_sigfun):
    '''一个样本的前向和后向计算'''

    global ann_yi
    global ann_delta
    global ann_w
    global ann_wj0
    global ann_y0
    global hidelevel_count
    global alllevel_count
    global learn_r
    global train_a
    global ann_oldw

    level=hidelevel_count
    #清空yi输出信号数组
    hidelevel=hidelevel_count
    alllevel=alllevel_count
    for i in xrange(0,alllevel):
        #第一维是层数，从0开始
        for j in xrange(0,n):
            #第二维是神经元
            ann_yi[i][j]=0.0
    ann_yi=np.array(ann_yi)
    yi=ann_yi

    #清空delta矩阵
    for i in xrange(0,hidelevel-1):
        for j in xrange(0,n):
            ann_delta[i][j]=0.0

    delta=ann_delta

```

```

#保留W的拷贝，以便下一次迭代
ann_oldw=copy.deepcopy(ann_w)
oldw=ann_oldw
#前向计算
if isdebug:print u"前向计算中..."
#对输入变量进行预处理

myo=np.array([])
for nowlevel in xrange(0,alllevel):
    #一层层向前计算
    #计算诱导局部域
    my_y=[]
    myy=yi[nowlevel-1]
    myw=ann_w[nowlevel-1]
    if nowlevel==0:
        #第一层隐藏层
        my_y=myx
        yi[nowlevel]=my_y
    elif nowlevel==(alllevel-1):
        #输出层
        my_y=o_func(yi[nowlevel-1,:len(myd)])
        yi[nowlevel,:len(myd)]=my_y
    elif nowlevel==(hidelvel-1):
        #最后一层输出层
        for i in xrange(0,len(myd)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel,:len(myd)]=my_y
    else:
        #中间隐藏层
        for i in xrange(0,len(myy)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel]=my_y

if isdebug:
    print u"*****本样本训练的输出矩阵*****"
    print yi
    print u"*****"

#计算误差与均方误差
#因为线性输出层为直接复制，所以取非线性隐藏输出层的结果
myo=yi[hidelvel-1][:len(myd)]
myo_end=yi[alllevel-1][:len(myd)]
mymse=get_e(myd,myo_end)

#反向计算
#输入层不需要计算delta,输出层不需要计算W
if isdebug:print u"反向计算中..."

```

```

#计算delta
for nowlevel in xrange(level-1,0,-1):
    if nowlevel==level-1:
        mydelta=delta[nowlevel]
        my_n=len(myd)
    else:
        mydelta=delta[nowlevel+1]
        my_n=n
    myw=ann_w[nowlevel]
    if nowlevel==level-1:
        #输出层
        mydelta=delta_sigfun(myo,myd,None,None,None,None,None)
    ## 
    elif nowlevel==level-2:
        #输出隐藏层的前一层，因为输出结果和前一层隐藏层的神经元数目可能存在
        不一致的情况，所以单独处理，传输相当于输出隐藏层的神经元数目的数据
        mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
        1,my_n,mydelta[:len(myd)],myw[:,len(myd)])
    else:
        mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
        1,my_n,mydelta,myw)

    delta[nowlevel][:my_n]=mydelta
#计算与更新权值w
for nowlevel in xrange(level-1,0,-1):
    #每个层的权值不一样
    if nowlevel==level-1:
        #输出层
        my_n=len(myd)
        mylearn_r=learn_r*0.8
        mytrain_a=train_a*1.8
    elif nowlevel==1:
        #输入层
        my_n=len(myd)
        mylearn_r=learn_r*0.9
        mytrain_a=train_a*0.8
    else:
        #其他层
        my_n=n
        mylearn_r=learn_r
        mytrain_a=train_a

    pre_level_myy=yi[nowlevel-1]
    pretrain_myww=oldw[nowlevel-1]
    pretrain_myw=pretrain_myww[:,my_n]

    #第二个调整参数
    temp_i=[]

```

```

        for i in xrange(0,n):
            temp_i.append([])
            for jj in xrange(0,my_n):
                temp_i[i].append(mylearn_r*delta[nowlevel,jj]*pre_
                level_myw[i])
        temp_rs2=np.array(temp_i)
        temp_rs1=mytrain_a*pretrain_myw
        #总调整参数
        temp_change=temp_rs1+temp_rs2
        my_ww=ann_w[nowlevel-1]
        my_ww[:,my_n]+=temp_change
    if isdebug:
        print "===="
        print u"***权值矩阵***"
        print ann_w
        print u"***梯度矩阵***"
        print delta
        print "===="
    return myse

```

经过 43 次训练，误差率降到了 0.02 以下，训练过程停止，如下所示：

```

.....
.....
-----开始第41次训练----- 误差为: 0.026790
-----开始第42次训练----- 误差为: 0.021292
-----开始第43次训练----- 误差为: 0.019350
训练成功，正在进行检验

```

最后，进行仿真测试。下面是一个样本值的仿真计算过程。

```

def simulate(myx,sigmoid_func,delta_sigfun):
    '''一个样本的仿真计算'''
    print u"仿真计算中"
    global ann_yi
    global ann_w
    global ann_wj0
    global ann_y0
    global hidelevel_count
    global alllevel_count
    global d
    global mylnww

    myd=d[0]
    myx=np.array(myx)
    n=len(myx)

    #清空yi输出信号数组
    hidelevel=hidelevel_count
    alllevel=alllevel_count
    for i in xrange(0,alllevel):
        #第一维是层数，从0开始

```

```

        for j in xrange(0,n):
            #第二维是神经元
            ann_yi[i][j]=0.0
    ann_yi=np.array(ann_yi)
    yi=ann_yi

    #前向计算
    myy=np.array([])
    for nowlevel in xrange(0,alllevel):
        #一层层向前计算
        #计算诱导局部域
        my_y=[]
        myy=yi[nowlevel-1]
        myw=ann_w[nowlevel-1]
        if nowlevel==0:
            #第一层隐藏层
            my_y=myx
            yi[nowlevel]=my_y
        elif nowlevel==(alllevel-1):
            #线性输出层
            my_y=o_func(yi[nowlevel-1,:len(myd)])
            yi[nowlevel,:len(myd)]=my_y
        elif nowlevel==(hidelvel-1):
            #最后一层隐藏输出层
            for i in xrange(0,len(myd)):
                temp_y=sigmoid_func(np.dot(myw[:,i],myy))
                my_y.append(temp_y)

            yi[nowlevel,:len(myd)]=my_y
        else:
            #中间隐藏层
            #中间隐藏层需要加上偏置
            for i in xrange(0,len(myy)):
                temp_y=sigmoid_func(np.dot(myw[:,i],myy))
                my_y.append(temp_y)
            yi[nowlevel]=my_y

    if isdebug:
        print "===="
        print u"***权值矩阵***"
        print ann_w
        print u"***输出矩阵***"
        print yi
        print "===="
    return yi[alllevel-1,:len(myd)]

```

为了验证效果，绘制数据拟合效果和误差曲线，如图 8-28 所示。图 8-28 的上部是拟合效果图，目标值和预测值很接近，下部是误差曲线，下降平滑。

(2) $0.6\sin(x)$ 函数神经网络拟合。下面尝试实现稍复杂数学函数的拟合，比如 $y=0.6\sin(x)$ 。这里的 Python 实现代码与前面相同，因此，只是在样本数据初始化代码段进行

了修改。示例如下：

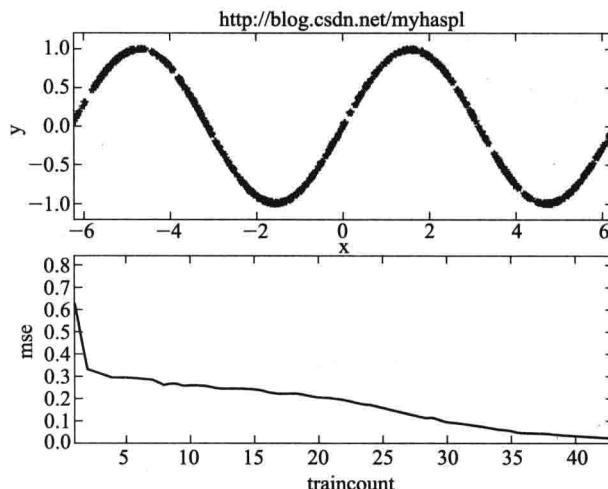


图 8-28 sin 函数神经网络拟合

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-2.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy

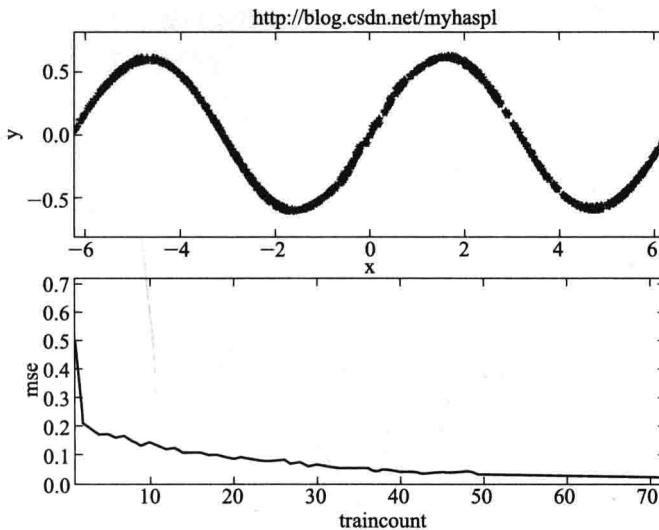
isdebug=False
#x和d样本初始化
train_x = []
d = []
for yb_i in xrange(0,500):
    train_x.append([np.random.rand()*4*np.pi-2*np.pi])
for yb_i in xrange(0,500):
    d.append(np.sin(train_x[yb_i])*0.6)
```

经过 71 次训练，神经网络的收敛目标达到，也就是说误差率达到了期望值。

```
.....
-----
-----开始第69次训练----- 误差为: 0.020464
-----开始第70次训练----- 误差为: 0.020673
-----开始第71次训练----- 误差为: 0.019350
训练成功，正在进行检验
```

图 8-29 为程序绘制的拟合效果和误差曲线图。

(3) $0.5\sin(x)+0.5\cos(x)$ 函数拟合。前两个例子使用了本书资源包所示的 Python 代码。下面使用 Neurolab 库实现对 $0.5\sin(x)+0.5\cos(x)$ 的拟合。代码如下：

图 8-29 $0.6\sin(x)$ 函数神经网络拟合（附彩图）

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#8-3.py
#拟合sin*0.5+cos*0.5
import neurolab as nl
import numpy as np
import matplotlib.pyplot as plt
isdebug=False

#x和d样本初始化
train_x = []
d= []
samplescount=1000
myrndsmp=np.random.rand(samplescount)
for yb_i in xrange(0,samplescount):
    train_x.append([myrndsmp[yb_i]*4*np.pi-2*np.pi])
for yb_i in xrange(0,samplescount):
    d.append(np.sin(train_x[yb_i])*0.5+np.cos(train_x[yb_i])*0.5)

myinput=np.array(train_x)
mytarget=np.array(d)

bpnet = nl.net.newff([-2*np.pi, 2*np.pi], [5, 1])
err = bpnet.train(myinput, mytarget, epochs=800, show=100, goal=0.02)

simd=[]
for xn in xrange(0,len(train_x)):
    simd.append(bpnet.sim([train_x[xn]])[0][0])

temp_x=[]
```

```

temp_y=simd
temp_d=[]
i=0
for mysamp in train_x:
    temp_x.append(mysamp[0])
    temp_d.append(d[i][0])
    i+=1
x_max=max(temp_x)
x_min=min(temp_x)
y_max=max(max(temp_y),max(d))+0.2
y_min=min(min(temp_y),min(d))-0.2

plt.xlabel(u"x")
plt.xlim(x_min, x_max)
plt.ylabel(u"y")
plt.ylim(y_min, y_max)

lp_x1 = temp_x
lp_x2 = temp_y
lp_d = temp_d
plt.plot(lp_x1, lp_x2, 'r*')
plt.plot(lp_x1,lp_d,'bo')
plt.show()

```

800 次训练后，拟合效果较好，如图 8-30 所示。

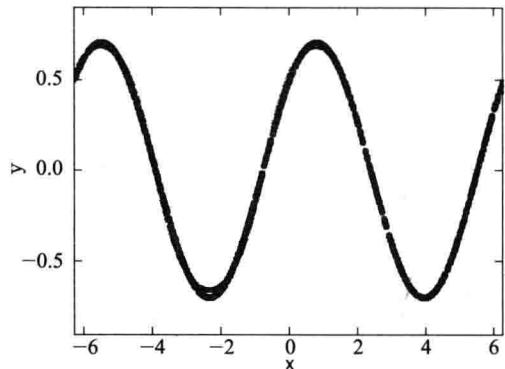


图 8-30 $0.5\sin(x)+0.5\cos(x)$ 的拟合

2. 钢包使用次数与容积模型拟合

表 8-3 是钢包使用次数与容积实测数据，以 x 为输入， y 为输出，在输入与输出数据之间可建立非线性关系。这里用神经网络建立数据拟合模型。

表 8-3 钢包使用次数与容积实测数据

使用次数 x	容积 y
2	106.42
3	108.2
4	109.58
5	109.5
7	110
8	109.93
10	110.49
11	110.59
14	110.6
15	110.9
16	110.7
18	111
19	111.2

下面尝试应用神经网络完成以上数据拟合任务，这里调用 Neurolab 库，用 Python 实现。

首先读取数据文件。代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-4.py
import numpy as np
import pylab as pl
import neurolab as nl

print u'正在处理中'

#x和d样本初始化
train_x = []
d = []
f = open("cubage.csv")
try:
    f_text = f.read()
finally:
    f.close()
x_text=f_text.split('\n')
for line_i in xrange(0,len(x_text)):
    line=x_text[line_i]
    if line_i>1 and len(line)>0:
        train_x.append([])
        hdata=line.split(',')
        train_x[line_i-2].append(float(hdata[0]))
        d.append([float(hdata[1])])
```

接着，生成样本数据，一般情况下，使用 tanh 的神经网络输出值不会超过 1，因此，设置调整系数，把输出值处理成 1 以内的小数。代码如下：

```
myinput=np.array(train_x)
mytarget=np.array(d)
mymax=np.max(d)
tz=(0.1**len(str(int(mymax))))*5
myinput=myinput
mytarget=tz*mytarget
```

最后进行训练，训练时可加入未知样本进行测试。

```
#对未知样本进行测试
myinputtest=[[6],[9],[17],[20]]
testsimd= bpnet.sim(myinputtest)
end_x=np.array(myinputtest)
testsimd/=tz
end_y=testsimd
```

经过 9 次训练，达到了训练目标，误差为 1.9713682268777952e-05。对神经网络输出值应用调整系数，将其输出值恢复到原有的数值范围。

程序生成了拟合效果图（如图 8-31 上部所示）及误差曲线图（如图 8-31 下部所示）。同时使用在样本空间中没有出现的部分测试数据 6、9、17、20 作为神经网络的输入，验证神经网络的训练效果和拟合效果。图中实心圆圈为未知测试数据，即：使用次数为 6、9、17、20 时容积的预测值，星号为样本数据。

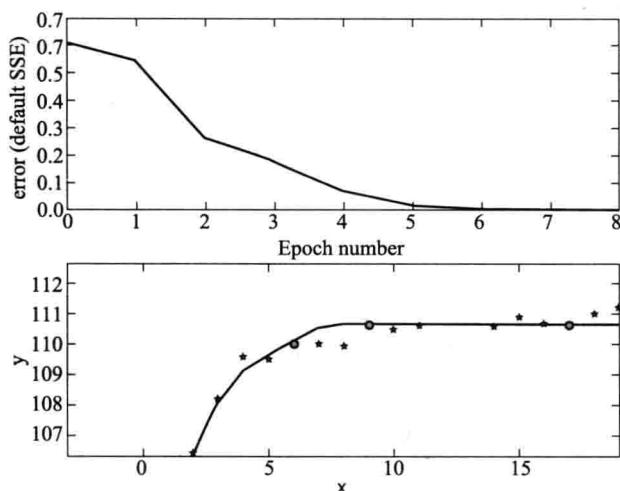


图 8-31 钢包使用次数与容积的神经网络拟合

从图 8-31 的拟合效果来看，样本数据非常接近神经网络拟合的曲线，说明曲线较好地反映了随着使用次数的增加，容积的增长趋势。此外，对在样本中没出现的钢包使用次数 6、9、17、20，与之相关的容积预测效果不错。

完整的 Python 代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-4.py
import numpy as np
import pylab as pl
import neurolab as nl
print u'正在处理中'
#x和d样本初始化
train_x = []
d = []
f = open("cubage.csv")
try:
    f_text = f.read()
finally:
    f.close()
x_text=f_text.split('\n')
for line_i in xrange(0,len(x_text)):
    line=x_text[line_i]
```

```

if line_i>1 and len(line)>0:
    train_x.append([])
    hdata=line.split(',')
    train_x[line_i-2].append(float(hdata[0]))
    d.append([float(hdata[1])])
myinput=np.array(train_x)
mytarget=np.array(d)
mymax=np.max(d)
tz=(0.1**len(str(int(mymax))))*5
myinput=myinput*tz
mytarget=tz*mytarget
netminmax=[0,np.max(myinput)]
print u'\n正在建立神经网络'
bpnet = nl.net.newff([netminmax], [5, 1])

print u'\n训练神经网络中...'
err = bpnet.train(myinput, mytarget, epochs=800, show=5, goal=0.0001)
if err[len(err)-1]>0.0001:
    print u'\n训练神经网络失败...\n'
else:
    print u'\n训练神经网络完毕'
    pl.subplot(211)
    pl.plot(err)
    pl.xlabel('Epoch number')
    pl.ylabel('error (default SSE)')
    #对样本进行测试
    simd= bpnet.sim(myinput)
    temp_x=myinput
    temp_d=mytarget
    simd/=tz
    temp_y=simd
    temp_d/=tz

#对未知样本进行测试
myinputtest=[[6],[9],[17],[20]]
testsimd= bpnet.sim(myinputtest)
end_x=np.array(myinputtest)
testsimd/=tz
end_y=testsimd

x_max=np.max(temp_x)
x_min=np.min(temp_x)-5
y_max=np.max(temp_y)+2
y_min=np.min(temp_y)

pl.subplot(212)
pl.xlabel(u"x")
pl.xlim(x_min, x_max)
pl.ylabel(u"y")
pl.ylim(y_min, y_max)

```

```

lp_x1 = temp_x
lp_x2=temp_y
lp_d = temp_d
pl.plot(lp_x1, lp_x2, 'g-')
pl.plot(end_x, end_y, 'ro')
pl.plot(lp_x1,lp_d,'b*')

```

8.2 线性滤波

8.2.1 WAV 声音文件

声音是由物体的机械振动而形成的。用鼓棒敲击鼓皮，于是鼓皮发生振动而发声；吹笛时笛腔内的空气柱发生振动而发声；把音频电流送入扬声器，扬声器的纸盆发生振动而发声。发生声音的振动源叫作“声源”，由声源发出的声音，必须通过媒质才能传送到我们的耳朵中。空气是最常见的媒质，如水、金属、木材等媒质都能传播声音。

WAV 声音文件为 Microsoft 公司开发的一种记录声音的文件格式，它符合 RIFF（Resource Interchange File Format）文件规范，音频格式未经过压缩，在音质方面不会出现失真的情况。它以指定频率采样，比如每秒采样 44100 次。在采集声音的振动状态时，它会使用模 / 数转换器（A/D）以每秒上万次的速率对声波进行采样，每一次采样都记录下了原始模拟声波在某一时刻的状态，采样的结果即为样本。将一串样本连接起来，就可以描述一段声波了。

8.2.2 线性滤波算法过程

神经网络既可以进行函数拟合，也能对波形数据进行拟合。下面将输入 x 作为函数的自变量，将神经网络输出的数据 y 作为函数 $f(x)$ 的输出，然后应用该拟合功能进行一个线性滤波，以实现去除周围环境的噪音，使说话声音更清楚。

下面以从含有音乐的语音中去除背景音乐为例进行讲解。具体算法过程如下：

(1) 读取该机器学习算法必需的两个素材文件：含有背景音乐的语音和部分背景音乐。细心的读者一定会问，既然已经有背景音乐这个文件，那么直接从语音的波形数据减去背景音乐，这样就完成了背景音乐的过滤。没错，但是不能直接减去源背景音乐。原因如下：

在对声波进行采样时，虽然同时对背景音乐和语音进行了采样，但采样过程很可能被周边的音源、采样音量等很多因素干扰，导致采样形成的背景音乐波形并不是源背景音乐的波形。如果直接使用源背景音乐进行减，将会导致滤波后的语音文件有杂音且部分失真。我们要做的是干净地将背景音乐剔除。

(2) 采样器先采样一小段背景音乐，然后再采集语音文件。其好处在于：生成了经过采样后的背景音乐样本，这些样本就是神经网络拟合训练样本中的目标输出值。

(3) 用神经网络进行训练，输入样本为源背景音乐中相当于采样背景音乐长度的波形数据，输出目标为采样器开始采集的小段背景音乐波形数据。

(4) 训练达到期望误差率或达到最大训练次数后，将源背景音乐作为未知样本数据送入

神经网络，其预测输出就是拟合后的采样背景音乐。

(5) 将混杂有背景音乐的语音文件波形数据直接减去拟合后的采样背景音乐，得到去除背景音乐的纯净语音。

自适应线性滤波器的工作原理也是如此：先采集一段背景噪声；然后用噪声数据进行拟合，之后在噪声数据和背景噪声之间建立了一种映射关系；最后，人可以通过该系统传送语音，说话周围环境产生的背景噪声将被过滤。

8.2.3 滤波 Python 实现

下面用 Python 实现上述步骤。

(1) 读取声音素材，代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-5.py
import numpy as np
import wave
import pylab as pl
import copy
print 'working...'

print "read wav data...."
err=[]
# 打开WAV文档
f = wave.open(r"speak.wav", "rb")
fo = wave.open(r"wait_jg.wav", "wb")
fi=wave.open(r"back.wav", "rb")
fend=wave.open(r"end_jg.wav", "wb")

# 读取波形数据
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)

fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)

#将波形数据转换为数组，并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
```

(2) 复制波形，在一个随机的基数基础上，将背景音乐的振幅（音量）稍微变动一下，并与语音合并。前面预留一段经过波形调整后的背景音乐，以供线性神经网络拟合用。

```
emptywdata=np.zeros(framerate, dtype=np.short)
```

```

new_wave_data=np.hstack((emptywdata, wave_data, wave_data, wave_data, wave_
data, wave_data, wave_data, wave_data))
wave_data =copy.deepcopy(new_wave_data)
nframes*=8
nframes+=framerate/2
temp_wavedata=np.hstack((fi_wave_data, fi_wave_data))[:len(new_wave_data)]
backrnd=np.random.rand(len(new_wave_data))*10-5
backbase=np.random.rand()*2+1
temp_wavedata=temp_wavedata*backbase+backrnd
new_wave_data=temp_wavedata+new_wave_data

new_wave_data=np.array(new_wave_data)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()

```

(3) 拟合数据，去除背景音乐。

```

jg_wave_data=copy.deepcopy(new_wave_data)
jg_temp_wavedata=np.hstack((fi_wave_data, fi_wave_data))[:len(new_wave_data)]
jg_temp_wavedata=jg_temp_wavedata[:len(new_wave_data)]*w[1]+w[0]
jg_wave_data=jg_wave_data-jg_temp_wavedata

for jg_i in xrange(0, len(jg_wave_data)):
    if abs(jg_wave_data[jg_i])<500:
        jg_wave_data[jg_i]=0
jg_wave_data[:framerate]=0

jg_wave_data =jg_wave_data.astype(wave_data.dtype)
jg_str_data=jg_wave_data.tostring()

print "save output wav...."
fend.setnchannels(nchannels)
fend.setframerate(framerate)
fend.setsampwidth(sampwidth)
fend.writeframes(jg_str_data)

```

完整的代码如下：

```

#!/usr/bin/env python
#-- coding: utf-8 --
#code:myhaspl@qq.com
#8-5.py
import numpy as np
import wave
import pylab as pl
import copy
print 'working...'

print "read wav data...."
err=[]
# 打开WAV文档
f = wave.open(r"speak.wav", "rb")

```

```

fo = wave.open(r"wait_jg.wav", "wb")
fi=wave.open(r"back.wav", "rb")
fend=wave.open(r"end_jg.wav", "wb")

# 读取波形数据
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)

fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)

#将波形数据转换为数组，并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)

#复制并将背景音乐的振幅(音量)在一个随机的基本基础上稍微变动后，与语音合并
#前面预留一段经过波形调整后的背景音乐，以供线性神经网络拟合用
emptywdata=np.zeros(framerate, dtype=np.short)
new_wave_data=np.hstack((emptywdata, wave_data, wave_data, wave_data,
wave_data, wave_data, wave_data, wave_data))
wave_data =copy.deepcopy(new_wave_data)
nframes*=8
nframes+=framerate/2
temp_wavedata=np.hstack((fi_wave_data, fi_wave_data))[:len(new_wave_data)]
backrnd=np.random.rand(len(new_wave_data))*10-5
backbase=np.random.rand()*2+1
temp_wavedata=temp_wavedata*backbase+backrnd
new_wave_data=temp_wavedata+new_wave_data

new_wave_data=np.array(new_wave_data)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
#写波形数据参数
print "save mix wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)

#线性逼近前段噪声
b=1
a0=5e-1
a=0.0
r=1.5
x=[]
d=[]

```

```

ii=0
for audio_i in xrange(0,framerate/2):
    if fi_wave_data[audio_i]!=0.:
        x.append([])
        x[ii].append(1)
        x[ii].append(fi_wave_data[audio_i])
        d.append(new_wave_data[audio_i])
        ii+=1
    if ii>100:
        break
x=np.array(x)
d=np.array(d)

w=np.random.rand(2)*np.mean(x) #np.array([b,0])
expect_e=15
maxtrycount=10000
mycount=0
def sgn(v):
    return v
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    a=a0/(1+float(mycount)/r)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)

while True:
    mye=0.
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye=np.sqrt(mye)
    mycount+=1
    err.append(mye)
    print u"第 %d 次调整后的权值: "%mycount
    print w
    print u"误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break

print "w:[%f,%f] "%(w[0],w[1])

#复制并除去背景声音
jg_wave_data=copy.deepcopy(new_wave_data)
jg_temp_wavedata=np.hstack((fi_wave_data,fi_wave_data))[:len(new_wave_data)]
jg_temp_wavedata=jg_temp_wavedata[:len(new_wave_data)]*w[1]+w[0]
jg_wave_data=jg_wave_data-jg_temp_wavedata

```

```

for jg_i in xrange(0, len(jg_wave_data)):
    if abs(jg_wave_data[jg_i])<500:
        jg_wave_data[jg_i]=0
    jg_wave_data[:framerate]=0

jg_wave_data = jg_wave_data.astype(wave_data.dtype)
jg_str_data=jg_wave_data.tostring()

print "save output wav...."
fend.setnchannels(nchannels)
fend.setframerate(framerate)
fend.setsampwidth(sampwidth)
fend.writeframes(jg_str_data)

# 绘制波形
time = np.arange(0, nframes) * (1.0 / framerate)
wave_data.shape = -1, 2
wave_data = wave_data.T

pl.subplot(321)
pl.plot(time, wave_data[0])
pl.subplot(322)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")

# 绘制波形
new_wave_data.shape = -1, 2
new_wave_data = new_wave_data.T

pl.subplot(323)
pl.plot(time,new_wave_data[0])
pl.subplot(324)
pl.plot(time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

# 绘制波形
jg_wave_data.shape = -1, 2
jg_wave_data = jg_wave_data.T

pl.subplot(325)
pl.plot(time,jg_wave_data[0])
pl.subplot(326)
pl.plot(time, jg_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

如图 8-32 所示是程序运行完毕后生成的效果图。最上面的两张图是无背景音乐语音文

件的两个声道的波形图，中间是混杂了背景音乐的语音文件，下面是经过线性滤波后生成的纯净语音文件。

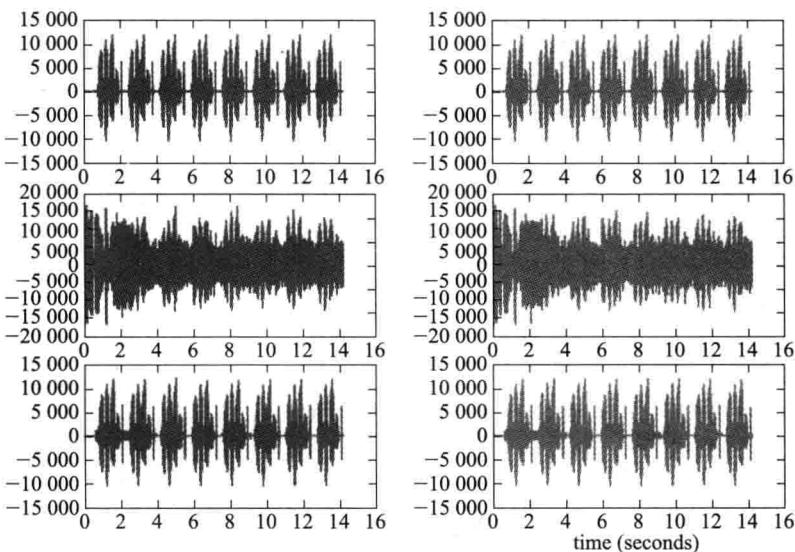


图 8-32 线性滤波效果

观察图 8-32 可看出，线性滤波的效果是不错的。读者可以从本书的资源包中下载相关程序，执行后，用音箱播放滤波后的语音，听听效果，是否还能听出背景音乐。

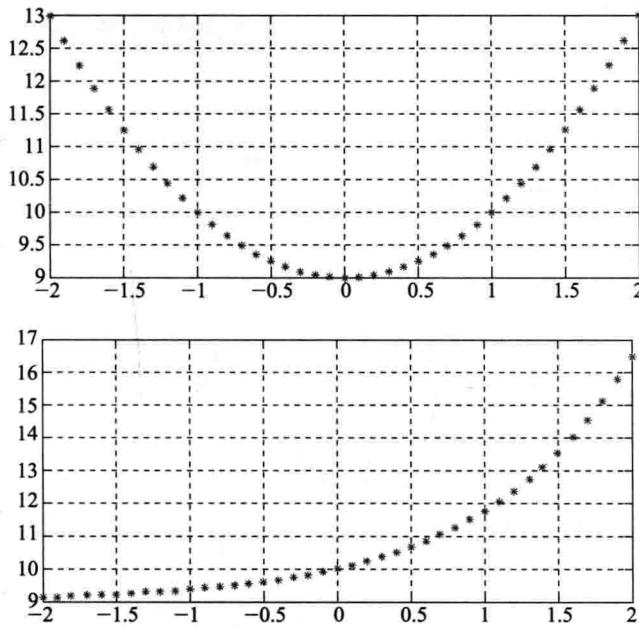
8.3 小结

本章主要介绍了使用回归方程模型和神经网络进行数据拟合的方式，重点讲解了回归方程估计及其散点图分析、神经网络对函数和数值的拟合、线性滤波去除语音背景噪声等。

数据拟合是通过理想的假设方程来拟合数据的，得到这个假设方程有两种方法：第一，可以通过观察样本数据点的分布来估计所属函数图像，在拟合后计算相关统计指标，估计拟合的效果；第二，以假设方程的自变量为输入样本，因变量为样本的目标输出，通过神经网络进行训练，以权值矩阵和众多神经元的激活函数等神经网络组件完成输入到输出的映射，其实质是建立一个更复杂的拟合模型实现数据拟合。

思考题

(1) 假设有两类数据，下面是这两类数据的分布散点图，从图像上分析拟合这些数据的函数方程。



(2) 编写 Python 代码实现多层感知器，拟合 $y=0.7\sin(x)+0.3\cos(x)$ 函数。

(3) 本章讲解了通过线性滤波去除背景音乐的方法。请尝试用非线性滤波解决这个问题，即：使用多层感知器对背景音乐进行拟合，然后将它从语音文件中去除。

第9章

图像识别案例

计算机视觉是一门研究如何使机器“看懂”图像的科学，即用摄像机和计算机代替人眼对目标进行识别、跟踪和测量等。首先，使用摄像机等设备采集画面，生成数字图像；然后，用计算机对图像进行分析，相当于人的大脑对眼睛采集的信息进行加工，得到所需的信息。

计算机视觉、图像处理、图像分析、机器人视觉、机器视觉等都是彼此紧密关联的学科，这些学科都属于人工智能的范畴，它们的研究目的都是使机器人（计算机）和人一样，能看到图像，理解图像，学习图像中包含的知识。机器学习是人工智能的核心技术，基于机器学习的图像算法应用于图像处理与分析领域，其作用相当于人类大脑加工和处理视觉神经反馈的图像。

机器学习算法加工数字图像的目的是提取所需的知识和信息，主要解决识别与监测、运动分析、场景重建、图像恢复等问题。其算法过程主要包括：图像获取、预处理、特征提取、加工分析、提取知识等。比如：OCR 算法首先通过电子设备扫描纸上打印的字符；然后检测暗、亮的模式来确定其形状；最后用智能识别方法将形状翻译成计算机文字。

9.1 图像边缘算法

9.1.1 数字图像基础

再次复习一下数字图像的知识。数字图像在计算机中保存为二维整数数组，数组中元素是二维图像中的像素，每个像素都用有限数值表示，对应于二维空间中一个特定的位置。图像是由二维像素点组成的矩阵，通常每个像素点由 3 个元素组成——红、绿、蓝，这 3 个基本分量可以组成高清的图像。也可以把图像上的每个像素点理解为 (x,y,z) 这样的一个点，这个点定义在三维空间，每一维分别代表红、绿、蓝分量。

假设将像素的顺序定义为：蓝、绿、红，那么就可以定义一个像素点为 $(blue,green,red)$ 。每个图像由大量的像素点组成，如果将这个像素点组成的矩阵定义为 $H \times W$ 大小 (H 为高， W 为宽)，那么就得到了一个 $H \times W \times 3$ 的矩阵（“3”表示像素点的数值由 3 个基本分量组成）。

OpenCV 作为图像算法库，对图像矩阵也是这么定义的。请提前安装好 OpenCV，本章大部分例子都需要它的 Python 邪定库。

假设图像矩阵的变量名为 img，现在要用 Python 实现一个蓝色分量为 200、绿色为 100、红色为 50 的像素点的定义，这个像素点位于图像的 300×150 处。代码如下：

```
img[300,150,0]=200
img[300,150,1]=100
img[300,150,2]=50
```

9.1.2 算法描述

算法的基本原理是：将当前像素与邻接的下部和右部的像素进行比较，如果相似，则将当前像素设置为黑色，否则设置为白色。如何判定像素相似呢？应用欧氏距离算法，将一个像素的 3 个色彩分量映射在三维空间中，如果 2 个像素点的欧氏距离小于某个常数的阈值，就认为它们相似。算法的最终效果如图 9-1 所示。图 9-1 中左图是原图像，右图是计算图像边缘的结果。

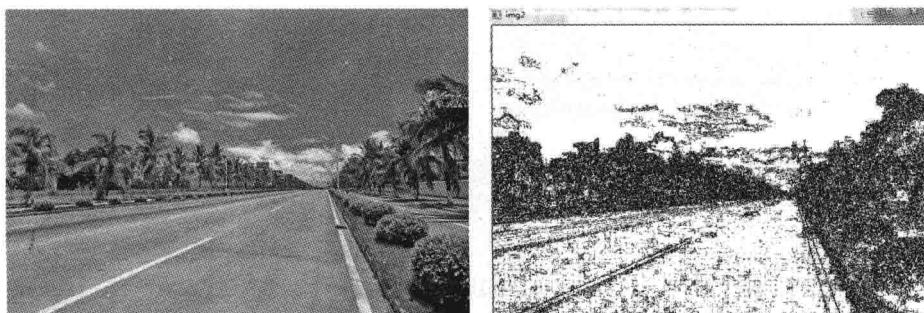


图 9-1 图像边缘

上面涉及的算法其关键是欧氏距离计算。下面用 Python 编写计算欧氏距离的函数。

```
def get_EuclideanDistance(x, y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))
```

完整的代码为：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-1.py
import cv2
import numpy as np

fn="test1.jpg"
def get_EuclideanDistance(x, y):
    myx=np.array(x)
```

```

myy=np.array(y)
return np.sqrt(np.sum((myx-myy)*(myx-myy)))

if __name__ == '__main__':
    print 'loading %s ...' % fn
    print 'working',
    myimg1 = cv2.imread(fn)
    w=myimg1.shape[1]
    h=myimg1.shape[0]
    sz1=w
    sz0=h
    #创建空白图像
    myimg2=np.zeros((sz0,sz1,3), np.uint8)
    #对比产生线条
    black=np.array([0,0,0])
    white=np.array([255,255,255])
    centercolor=np.array([125,125,125])
    for y in xrange(0,sz0-1):
        for x in xrange(0,sz1-1):
            mydown=myimg1[y+1,x,:]
            myright=myimg1[y,x+1,:]

            myhere=myimg1[y,x,:]
            lmyhere=myhere
            lmyright=myright
            lmydown=mydown
            if get_EuclideanDistance(lmyhere,lmydown)>16 and get_EuclideanDistance(lmyhere,lmyright)>16:
                myimg2[y,x,:]=black
            elif get_EuclideanDistance(lmyhere,lmydown)<=16 and get_EuclideanDistance(lmyhere,lmyright)<=16:
                myimg2[y,x,:]=white
            else:
                myimg2[y,x,:]=centercolor
            print '.',
    cv2.namedWindow('img2')
    cv2.imshow('img2', myimg2)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

9.2 图像匹配

图像匹配算法是基于像素的比较和计算来实现的方法。如图 9-2 所示是美国的 X-47B，它是人类历史上第一架无需人工干预、完全由计算机智能操纵的无人驾驶飞机。现在以其在航母起飞的图像为例讲解本节内容。



图 9-2 X-47B 航母起飞

如图 9-3 所示为图 9-2 的两张局部切片图。我们的任务是找到两张切片图在图 9-2 中的位置，并将它们标注出来。

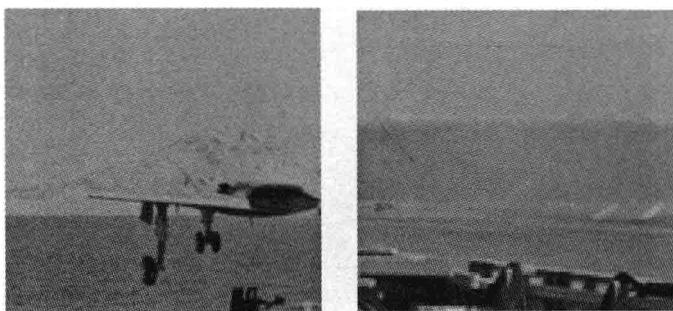


图 9-3 两张 X-47B 起飞画面切片图

9.2.1 差分矩阵求和

差分算法的核心在于差分矩阵，实质为差异矩阵，计算公式很简单：

$$\text{差分矩阵} = \text{图像 A 矩阵数据} - \text{图像 B 矩阵数据}$$

算法过程是：首先，计算两个图像的矩阵数据之间差异分析图像的相似性；然后，设置一个阈值进行比较，如果差分矩阵的所有元素之和在阈值以内，则表示这两张图像是相似的，且描述了同一物体。另外，它要求两个图像的大小相同，大小处理对于计算机来说不成问题，改变图像尺寸的算法已非常成熟，实现起来很方便。

编写程序实现这个算法的基本思路为：将图 9-3 所示的切片图在图 9-2 中进行移动，并计算两个图像的差分矩阵，如果差分矩阵的所有元素之和小于 1，则认为找到了切片图在图像中的位置。实现该算法的 Python 代码如下：

```
def showpiclocation(img, findimg):
```

```

#定位图像
w=img.shape[1]
h=img.shape[0]
fw=findimg.shape[1]
fh=findimg.shape[0]
findpt=None
for now_h in xrange(0,h-fh):
    for now_w in xrange(0,w-fw):
        comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-findimg
        if np.sum(comp_tz)<1:
            findpt=now_w,now_h
            print ".",
if findpt!=None:
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (255,0,0))
return img

```

如图 9-4 所示为算法效果图，看上去识别效果不错。

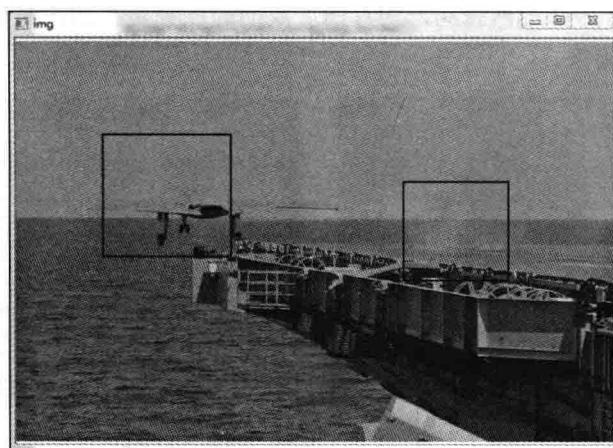


图 9-4 切片识别效果图

完整的 Python 代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-2.py
#简单定位图像

import cv2
import numpy as np
print 'loading ...'
def showpiclocation(img,findimg):
    #定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]

```

```

fh=findimg.shape[0]
findpt=None
for now_h in xrange(0,h-fh):
    for now_w in xrange(0,w-fw):
        comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-findimg
        if np.sum(comp_tz)<1:
            findpt=now_w,now_h
        print ".",
    if findpt!=None:
        cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (255,0,0))
return img

fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)

myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

9.2.2 差分矩阵均值

刚才小试牛刀，通过对差分矩阵中所有元素求和完成了匹配，效果不错。当数字图像质量较差时，则需要计算差分矩阵的均值，并为均值设一个适当的阈值。

下面仍用 Python 编写算法，在目标图中加上少量（50000个）不同颜色的噪声点，从而测试算法在弱噪声环境下的有效性。因为图像质量不高，存在很多噪声点，所以要将差分矩阵均值的阈值设置得稍大一点，这里设为 20。通常来说，阈值为 10~200，阈值越大，能容忍的噪声点越多。但如果阈值超过 200，最好使用下一节介绍的欧氏距离算法。如图 9-5 所示是算法应用效果。

Python 代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-3.py
#少量噪声定位图像

import cv2
import numpy as np

print 'loading ...'

```

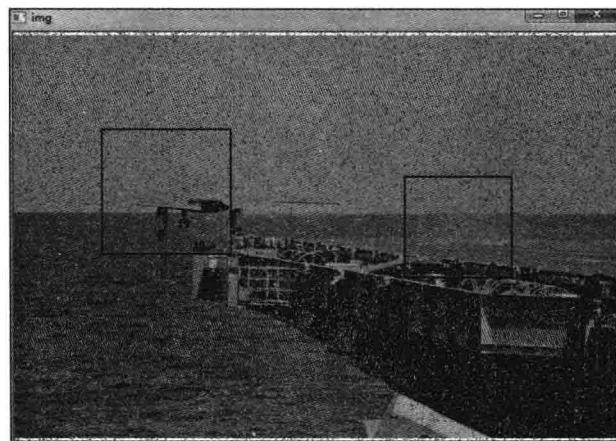


图 9-5 弱噪声切片识别效果图（附彩图）

```

def showpiclocation(img, findimg):
    #定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    findpt=None
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-findimg
            if abs(np.mean(comp_tz))<20:
                findpt=now_w,now_h
                print "ok"
            print ".",
    if findpt!=None:
        cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (0,0,255))
    return img

def addnoise(img):
    coutn=50000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)

```

```

addnoise(myimg)
myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

可见，在弱噪声的情况下，差分算法仍然具有很好的匹配效果。

9.2.3 欧氏距离匹配

1. 强噪声图像匹配

对强噪声环境下的图像进行匹配时，欧氏距离匹配方法相对于以上两种方法会有更好的效果。

仍以上面的图像为目标进行讲解。首先，在目标图像中加上更多的（500000个）不同颜色的噪声点。代码如下：

```

def addnoise(img):
    coutn=500000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

```

生成强噪声环境下的图像，如图 9-6 所示。

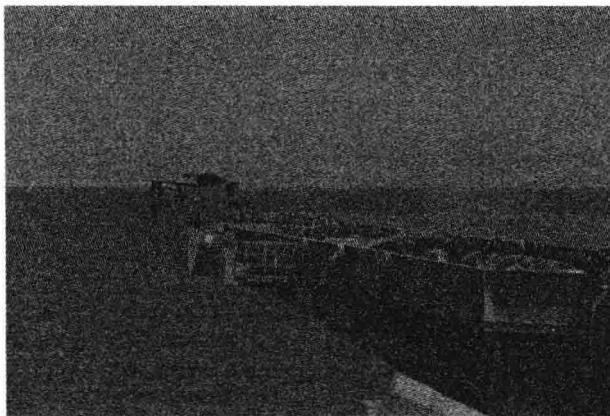


图 9-6 强噪声图像（附彩图）

现在要应用欧氏距离对如图 9-6 所示的目标图完成匹配。其核心算法为：设图像矩阵有 n 个元素，用 n 个元素值 (x_1, x_2, \dots, x_n) 组成该图像的特征组，特征组形成了 n 维空间，特征组中的特征码构成每一维的数值。在 n 维空间下，两个图像矩阵各形成了一个点，然后计算

这两个点之间的距离，距离最小者为最匹配的图像。

如图 9-7 所示是算法应用的效果图，欧氏距离算法成功地找到了匹配位置。在图像如此模糊的情况下，人用肉眼都很难进行图像匹配。由此可见，机器学习算法在某些方面胜过人类的“智能”。



图 9-7 强噪声切片识别效果图（附彩图）

完整的 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-4.py
#大量噪声定位图像

import cv2
import numpy as np

print 'http://blog.csdn.net/myhaspl'
print 'myhaspl@qq.com'
print
print 'loading ...'

def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,findimg,h,fh,w,fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            if
```

```

my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
my_findimg=findimg
dis=get_EuclideanDistance(my_img,my_findimg)
if dis<minds:
    mincb_h=now_h
    mincb_w=now_w
    minds=dis
print ".",
findpt=mincb_w,mincb_h
cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(0,0,255))
return img

def showpiclocation(img,findimg):
    #定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    return findpic(img,findimg,h,fh,w,fw)

def addnoise(img):
    coutn=500000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
addnoise(myimg)
myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow("img")
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

2. 变形图像匹配

欧氏距离方法不仅对强噪声图像匹配有效，而且对变形后的图像匹配效果也不错。如图9-8所示就是应用欧氏距离方法对有倾斜角度的图像进行匹配的效果。

算法原理前面已经解说过，在此不重复。相关的Python代码实现如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-

```



图 9-8 倾斜图像匹配效果图

```
#code:myhaspl@qq.com
#9-5.py
#图像倾斜后定位图像

import cv2
import numpy as np

print 'loading ...'

def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,findimg,h,fh,w,fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
            my_findimg=findimg
            dis=get_EuclideanDistance(my_img,my_findimg)
            if dis<minds:
                mincb_h=now_h
                mincb_w=now_w
                minds=dis
    print ".",
    findpt=[mincb_w,mincb_h]
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (0,0,255))
    return img
```

```

def showpiclocation(img, findimg):
    #定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    return findpic(img, findimg, h, fh, w, fw)

fn='pictestxz.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)

myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

欧氏距离对弱噪声环境下的变形图像仍有不错的效果，如图 9-9 所示。

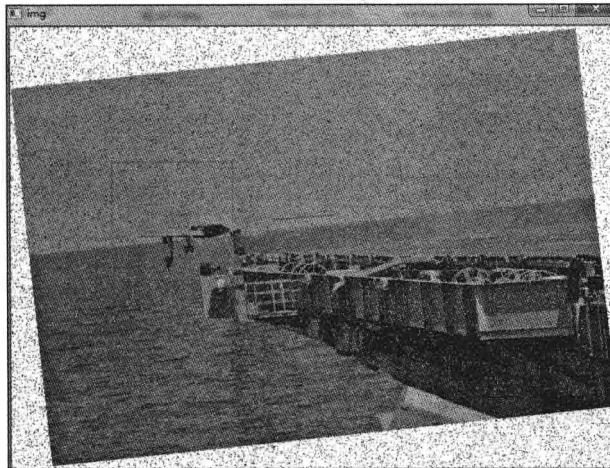


图 9-9 弱噪声变形图像匹配

该算法的 Python 实现代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-6.py
#图像倾斜后加噪声点，定位图像
import cv2

```

```

import numpy as np

print 'loading ...'
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,findimg,h,fh,w, fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
            my_findimg=findimg
            dis=get_EuclideanDistance(my_img,my_findimg)
            if dis<minds:
                mincb_h=now_h
                mincb_w=now_w
                minds=dis
        print ".",
    findpt=[mincb_w,mincb_h]
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (0,0,255))
    return img

def showpiclocation(img,findimg):
    #定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    return findpic(img,findimg,h,fh,w, fw)

def addnoise(img):
    coutn=50000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

fn='pictestxz.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
addnoise(myimg)

```

```

myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

9.3 图像分类

近年来随着多媒体技术的发展，图像分类技术受到了普遍的关注，目前采用的方法以机器学习算法为主。图像分类利用计算机对图像进行分析，根据图像信息的不同特征，将不同类别的图像区分开来。

图像分类的算法过程如下：

- (1) 准备样本图像。样本图像的要求是：能代表所属类别中尽可能多的图像。
- (2) 提取每个样本的特征后，形成类别特征码。
- (3) 应用机器学习算法对类别特征码进行学习，提取特征码包含的图像知识。
- (4) 判断未知图像所属类别。

9.3.1 余弦相似度

余弦相似度通过测量两个向量内积空间的夹角的余弦值来度量它们之间的相似性，尤其适用于任何维度的向量比较中，因此属于高维空间应用较多的机器学习算法。通常来说，数字图像包含的特征码较多，而这些特征组就属于高维空间，这正是余弦相似度算法应用的范围，算法将每个图像的特征组转化为高维空间的向量，两个向量之间的角度之余弦值可用于确定两个向量是否大致指向相同的方向。

在图像分类中应用余弦相似度算法的关键在于：计算这些代表每个图像特征的向量的内积空间的夹角余弦值，从而度量图像之间的相似性。对于相似性的衡量标准有以下两种：

- 为相似性设置一个阈值。在这个阈值以内的都属于同一类别图像。这种标准可以将图像划分为多种类型，例如：高楼不但属于城市美景，而且属于写字楼景观。
- 选择与样本向量的余弦相似度最接近 1 的图像为该类别图像。这种标准只能将图像划分为一种类别。

1. 算法描述

下面针对第二种衡量标准讲解余弦相似度算法。

特征提取一直是图像处理和计算机视觉研究领域中一个值得探讨的问题，在计算机科学、医疗辅助诊断、军事、工业测量等众多领域都广泛采用这一技术，尤其是在计算机视觉和模式识别的研究中。如何准确定位和提取关键特征往往是首先需要解决的问题之一，是提高识别率等问题的重要前期准备和关键因素。目前图像特征提取算法较多，不同的算法适应于不同的图像分析任务。

本节讲述的算法基本原理是：把图像上的点分为不同的子集，这些子集往往属于孤立的点、连续的曲线或者连续的区域。将这些点按区域组成子集，提取子集的特征后，将每个子集的特征作为图像的一个特征项来进行计算。

(1) 样本特征。设图像分为 m 个区域，每个区域有 n 个像素，每个像素在图像矩阵中以红、绿、蓝三色来表示，且区域特征计算方式为：

$$\text{区域特征码} = \left(\frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的红色分量值}}{n}, \frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的绿色分量值}}{n}, \frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的蓝色分量值}}{n} \right)$$

那么，样本特征码矩阵为 $3 \times m$ ，即共 3 行 m 列，这 3 行分别代表红、绿、蓝三个分量，每列为各分量的区域特征码。

(2) 类别特征。这里为每个类别准备了 3 个样本，类别特征的计算方式为：

$$\text{类别特征码} = \left(\frac{\sum_{i \in 3} \text{样本 } i \text{ 的红色分量特征值}}{3}, \frac{\sum_{i \in 3} \text{样本 } i \text{ 的绿色分量特征值}}{3}, \frac{\sum_{i \in 3} \text{样本 } i \text{ 的蓝色分量特征值}}{3} \right)$$

其中，样本 i 是属于该类别的样本。

在计算出类别特征后，算法对样本学习完毕。当有未知图像需要分类时，首先计算其图像的样本特征，然后将样本特征和类别特征映射为高维空间的向量，最后计算这两个向量的余弦相似度，选择余弦相似度最大的类别为未知图像对应的类别。

2. 算法应用

下面以风景图像分类为例说明余弦相似度的应用。为每个类别各准备 3 个样本图像，提取类别特征码，然后将如图 9-10~图 9-12 所示的 3 个待分类图像划分到蓝天风景、树林风景、瀑布风景这 3 个分类中。

3. Python 实现

(1) 将图 9-10~图 9-12 分别从上到下、从左到右分割成若干块状区域，对每块区域的图像像素特征进行提取后，形成这 3 个待分类图像的特征码。下面的代码所示的函数 `readpic` 定义了分割并提取特征码的操作，以待分类图像为参数调用该函数，完成特征码计算。



图 9-10 蓝天风景



图 9-11 树林风景



图 9-12 瀑布风景

```
def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnmimg,(800,600))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
    alltz.append([])
    alltz.append([])
    alltz.append([])
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz[0].append(btz)
            alltz[1].append(gtz)
            alltz[2].append(rtz)
    return alltz
```

(2) 计算类别特征码。通过每个类别所有样本的区域特征的平均值，提取类别特征。代码如下：

```
#读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    b_tz=np.array([0,0,0])
    g_tz=np.array([0,0,0])
    r_tz=np.array([0,0,0])
    mytz=np.zeros((3,w_fg*h_fg))
```

```

for jj in xrange(1,3):
    fn='p'+str(ii)+ '-' +str(jj) +'.png'
    tmptz=readpic(fn)
    mytz+=np.array(tmptz)
mytz/=3
train_x.append(mytz[0].tolist() + mytz[1].tolist() + mytz[2].tolist())

```

(3) 计算待分类图像的特征码与每个类别特征码之间的余弦距离, 距离最大者为图像所属分类。下面的代码计算了 ptest3.png 图像与每个类别特征码的余弦相似度。

```

fn='ptest3.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist() + testtz[1].tolist() + testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s属于第%d类'%(fn,nowi+1)

```

完整的 Python 代码如下:

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-7.py
#余弦距离识别图像类型
import numpy as np
import cv2

print u'正在处理中'
w_fg=20
h_fg=15
picflag=3
def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(800,600))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
    alltz.append([])
    alltz.append([])
    alltz.append([])
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]

```

```

r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
btz=np.mean(b)
gtz=np.mean(g)
rtz=np.mean(r)
alltz[0].append(btz)
alltz[1].append(gtz)
alltz[2].append(rtz)
return alltz

def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)

#x和d样本初始化
train_x = []
d = []

#读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    b_tz=np.array([0,0,0])
    g_tz=np.array([0,0,0])
    r_tz=np.array([0,0,0])
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,3):
        fn='p'+str(ii)+ '-' + str(jj) + '.png'
        tmptz=readpic(fn)
        mytz+=np.array(tmptz)
    mytz/=3
    train_x.append(mytz[0].tolist() + mytz[1].tolist() + mytz[2].tolist())

fn='ptest3.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist() + testtz[1].tolist() + testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s属于第%d类'%(fn,nowi+1)

fn='ptest1.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist() + testtz[1].tolist() + testtz[2].tolist()

```

```

maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s属于第%d类'%(fn,nowi+1)

fn='ptest2.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s属于第%d类'%(fn,nowi+1)

```

运行上述代码，观察下面的运行结果，可见对算法任务中列举的图像识别效果不错。

```

正在处理中
ptest3.png属于第3类
ptest1.png属于第1类
ptest2.png属于第2类

```

本节中每个类别仅使用了 3 个样本，基于余弦相似度的算法在样本量较小的情况下，效果其实不是最佳的。例如：在测试图像中加入另一张测试图，要用基于余弦相似度算法将它分类，那会怎样？修改 9-7.py，代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-8.py
#余弦距离识别图像类型，有一张图像不能正确识别
import numpy as np
import cv2
print u'正在处理中'
.....
.....
fn='ptest22.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim

```

```

nowi=i
print u'%s属于第%d类'%(fn,nowi+1)

```

程序运行结果如下：

```

正在处理中
ptest3.png属于第3类
ptest1.png属于第1类
ptest2.png属于第2类
ptest22.png属于第3类

```

从上述结果看，图像 ptest22.png（如图 9-13 所示）被错误地分到了第三类，实际它属于第二类树林风景图像。



图 9-13 待分类图像

9.3.2 PCA 图像特征提取算法

PCA 算法基于变量协方差矩阵对信息进行压缩和处理，通常用于数据降维，可将它用于图像矩阵降维，以降维后的矩阵为基础提取图像特征。当提取的图像特征维度比较高时，为了简化计算量以及储存空间，需要对这些高维数据进行一定程度上的降维，并尽量保证数据不失真。此外，PCA 算法还可应用于图像矩阵，它能找到变化大的维，去除掉那些变化不大的维，这样能更有效地提取图像明显特征，便于后期识别算法并进一步加工，因为图像特征组含有的不明显的特征值将会影响识别的精度。

应用 PCA 降维技术，对上节讲述的图像特征码算法进行改进，返回图像特征码。下面是用 Python 实现的基于 PCA 的图像特征码算法。

```

def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20

```

```

h_interval=h/10
alltz=[]
for now_h in xrange(0,h,h_interval):
    for now_w in xrange(0,w,w_interval):
        b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
        g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
        r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
        btz=np.mean(b)
        gtz=np.mean(g)
        rtz=np.mean(r)
        alltz.append([btz,gtz,rtz])
result_alltz=np.array(alltz).T
pca = mlpv.PCA()
pca.learn(result_alltz)
result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
result_alltz = result_alltz.reshape(len(result_alltz))
return result_alltz

```

下一节的神经网络与 SVM 图像分类算法将基于本节描述的 PCA 技术提取图像特征码，然后进一步分析和计算，得到图像所属类别。

9.3.3 基于神经网络的图像分类

基于神经网络的图像分类算法比余弦相似度分类算法的普适性更好，准确率更高。

1. 算法描述

神经网络图像分类算法首先通过 PCA 技术提取样本图像特征码与待分类图像特征码，然后将特征码送入神经网络进行训练，让神经网络学习每个类别图像的特征，最后将未知类别图像送入神经网络，自动识别它的类型。其步骤如下：

- (1) 基于 PCA 技术提取每个样本的图像特征码。
- (2) 根据样本特征码生成输入项，根据样本所属类别生成对应的输出项。
- (3) 将输入与输出项送入非线性神经网络训练。
- (4) 基于 PCA 技术生成待分类图像的特征码。
- (5) 将待分类图像的特征码送入神经网络仿真测试，根据神经网络输出项判断其所属类别。

2. 输出目标设计

神经网络的输出目标以及输出函数的设计应本着灵活实用的原则。在本例中，神经网络的输出值为 3 个图像类别，用数字 1~3 来表示，但不能直接将数字作为目标输出值。常用的表示方式有以下几种。

- 将数字转换为 1 以内的小数。比如：乘以输出值的最大数的倒数进行调整等。
- 按照二进制编码的思路，将其设计为如下形式：

1:[0,0,1]

2:[0,1,0]

3:[0,1,1]

或者，设计为以下格式（本例采用这种格式）：

1:[0,0,1]

2:[0,1,0]

3:[1,0,0]

输出函数的设计原则是：将输出目标值转化为实际样本的输出值格式。本例中将其定义为：

神经网络输出值 = 输出目标值的最大值所在的数组索引

3. Python 实现

下面来看看图像特征码计算，代码如下：

```
def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20
    h_interval=h/10
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlpv.PCA()
    pca.learn(result_alltz)
    result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
    result_alltz =result_alltz.reshape(len(result_alltz))
    return result_alltz
```

接着是输入与输出项初始化。代码如下：

```
#x和d样本初始化
train_x = []
d = []
sp_d = []
sp_d.append([0,0,1])
sp_d.append([0,1,0])
sp_d.append([1,0,0])
#读取图片
for ii in xrange(1,4):
    for jj in xrange(1,4):
```

```

fn='p'+str(ii)+ '-' +str(jj) +'.png'
pictz=readpic(fn)
train_x.append(pictz)
d.append(sp_d[ii-1])
myinput=np.array(train_x)
mytarget=np.array(d)

```

下面来看看训练效果。误差曲线如图 9-14 所示。

程序运行后，识别效果如下：

```

训练神经网络完毕
对样本进行测试
[1, 1, 1, 2, 2, 2, 3, 3, 3]
进行仿真
====ptest3.png====
[[ 0.96680714  0.00471284 -0.04523491]]
[3]
====ptest1.png====
[[ 0.10858063 -0.00820875  0.95785349]]
[1]
====ptest2.png====
[[ 0.01738297  0.99945777 -0.01017012]]
[2]
====ptest21.png====
[[ 0.95422417  0.00308943  0.12213834]]
[3]
====ptest22.png====
[[ -0.26776152  0.99953727 -0.12122857]]
[2]

```

最后几行表明，无法被余弦相似度正确分类的 ptest22.png 被神经网络分类成功，但 ptest21.png（如图 9-15 所示）被错误分类。因为神经网络与 SVM 的不同之处在于，神经网络训练需要更多的样本进行训练（本例仅使用 3 个样本），否则不一定能取得更好的识别效果。

完整的 Python 代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-8.py
#PCA加上人工神经网络识别图像类型
import numpy as np
import pylab as pl
import neurolab as nl
import cv2
import mlpy
print u'正在处理中'
def getresult(simjg):
    jg=[]

```

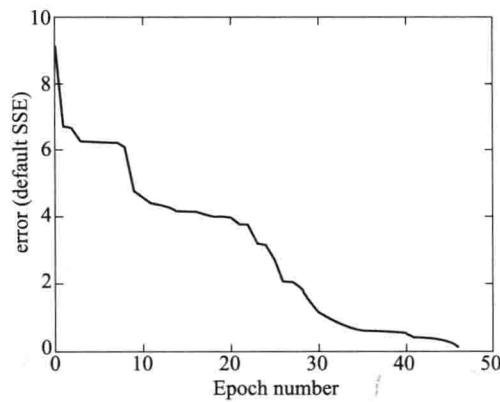


图 9-14 误差曲线



图 9-15 错误分类的图像

```

for j in xrange(0,len(simjg)):
    maxjg=-2
    nowii=0
    for i in xrange(0,len(simjg[0])):
        if simjg[j][i]>maxjg:
            maxjg=simjg[j][i]
            nowii=i
    jg.append(len(simjg[0])-nowii)
return jg
def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20
    h_interval=h/10
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlpv.PCA()
    pca.learn(result_alltz)
    result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
    result_alltz =result_alltz.reshape(len(result_alltz))
    return result_alltz
#x和d样本初始化
train_x = []
d=[]
sp_d=[]
sp_d.append([0,0,1])
sp_d.append([0,1,0])
sp_d.append([1,0,0])
#读取图像
for ii in xrange(1,4):
    for jj in xrange(1,4):
        fn='p'+str(ii)+ '-' +str(jj) +'.png'
        pictz=readpic(fn)
        train_x.append(pictz)
        d.append(sp_d[ii-1])
myinput=np.array(train_x)
mytarget=np.array(d)
mymax=np.max(myinput)

```

```

netminmax=[]
for i in xrange(0,len(myinput[0])):
    netminmax.append([0,mymax])

print u'\n正在建立神经网络'
bpnet = nl.net.newff(netminmax, [5, 3])

print u'\n训练神经网络中...'
err = bpnet.train(myinput, mytarget, epochs=800, show=5, goal=0.2)
if err[len(err)-1]>0.4:
    print u'\n训练神经网络失败...\n'
else:
    print u'\n训练神经网络完毕'
    pl.subplot(111)
    pl.plot(err)
    pl.xlabel('Epoch number')
    pl.ylabel('error (default SSE)')
    print u"对样本进行测试"
    simd= bpnet.sim(myinput)
    mysimd=getresult(simd)
    print mysimd
    print u"进行仿真"
    testpictz=np.array([readpic('ptest3.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "====ptest3.png===="
    print simtest
    print mysimtest
    testpictz=np.array([readpic('ptest1.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "====ptest1.png===="
    print simtest
    print mysimtest
    testpictz=np.array([readpic('ptest2.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "====ptest2.png===="
    print simtest
    print mysimtest
    testpictz=np.array([readpic('ptest21.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "====ptest21.png===="
    print simtest
    print mysimtest
    testpictz=np.array([readpic('ptest22.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "====ptest22.png===="

```

```
print simtest
print mysimtest
pl.show()
```

9.3.4 基于 SVM 的图像分类

在样本量少的情况下，SVM 分类的效果是最佳的。SVM 算法相比神经网络的优势在于：只需要少量样本，就能达到较高的识别精度。

1. 算法描述

SVM 图像分类算法首先通过 PCA 技术提取样本图像特征码与待分类图像特征码，然后将特征码送入 SVM 进行训练，学习每个类别图像的特征，最后将未知类别图像送入 SVM 仿真测试，自动识别它的类型。步骤如下：

- (1) 基于 PCA 技术提取每个样本的图像特征码。
- (2) 根据样本特征码生成输入项，根据样本所属类别生成对应的输出项。
- (3) 将输入与输出项送入 SVM 训练。
- (4) 基于 PCA 技术生成待分类图像的特征码。
- (5) 将待分类图像的特征码送入 SVM 仿真测试，根据 SVM 输出项判断其所属类别。

其中，SVM 的输出目标可以直接使用类别序号（在本例中为数字 1~3）。

2. Python 实现

- (1) 输入与输出项的初始化。代码如下：

```
#x和d样本初始化
train_x = []
d = []
#读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x = []
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,4):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        tmptz=readpic(fn)
        train_x.append(tmptz.tolist())
        d.append(ii)
```

- (2) SVM 训练与仿真训练。代码如下：

```
x=np.array(train_x)
y=np.array(d)
svm = mlpv.LibSvm(svm_type='c_svc', kernel_type='poly',gamma=50)
svm.learn(x, y)
print svm.pred(x)
```

- (3) 识别效果。如下所示：

```
正在处理中
[ 1.  1.  1.  2.  2.  2.  3.  3.  3.]
```

ptest3.png属于第3类
 ptest1.png属于第1类
 ptest2.png属于第2类
 ptest21.png属于第2类
 ptest22.png属于第2类

相对神经网络而言，在每个类别仅有3个样本的情况下，所有的测试图像得到了正确的分类。

以下是完整的代码：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-9.py
#PCA加上SVM识别图像类型
import numpy as np
import cv2
import mlp
print u'正在处理中'
w_fg=10
h_fg=5
picflag=3
def readpic(fn):
    #返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (400,200))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlp.PCA()
    pca.learn(result_alltz)
    result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
    result_alltz =result_alltz.reshape(len(result_alltz))
    return result_alltz

#x和d样本初始化
train_x = []
d = []
#读取图像，提取每类图像的特征
```

```

for ii in xrange(1,picflag+1):
    smp_x=[]
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,4):
        fn='p'+str(ii)+ '-' +str(jj) +'.png'
        tmptz=readpic(fn)
        train_x.append(tmptz.tolist())
        d.append(ii)
    x=np.array(train_x)
    y=np.array(d)
    svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=50)
    svm.learn(x, y)
    print svm.pred(x)
    fn='ptest3.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s属于第%d类'%(fn,nowi)
    fn='ptest1.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s属于第%d类'%(fn,nowi)
    fn='ptest2.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s属于第%d类'%(fn,nowi)
    fn='ptest21.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s属于第%d类'%(fn,nowi)
    fn='ptest22.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s属于第%d类'%(fn,nowi)

```

9.4 人脸辨识

生物特征识别技术，是指通过生物体（一般特指人）本身的生物特征来区分生物体个体。生物特征识别技术所研究的相关特征包括脸、指纹、手掌纹、虹膜、视网膜、声音、体形、个人习惯等。相应的识别技术有人脸识别、指纹识别、掌纹识别、虹膜识别、视网膜识别、语音识别、体形识别、键盘敲击识别、签字识别等。

人脸识别属于生物特征识别技术中的一种，指利用分析比较人脸视觉特征信息进行身份鉴别的计算机技术。

9.4.1 人脸定位

在一张图像或一段视频中定位人脸的技术目前已比较成熟，可调用 OpenCV 提供的接口

来完成。相应的 Python 代码如下：

```
def findface(image):
    #人脸识别，获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade =
        cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.015,
2, cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))
    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result
```

上面算法的原理是：首先将图像转换为灰度图，然后加载 OpenCV 提供的面部特征库，接着调用 HaarDetectObjects 找到人脸的位置，最后将定位的结果赋值给 result 数据并返回。

下面以如图 9-16 所示的《机械公敌》的电影海报为例来应用算法。

实现该算法的完整 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-10.py
#人脸定位

import cv2
import cv2.cv as cv
print 'loading ...'

#请在本程序运行前检查opencv的目录是否为下面的OPCV_PATH值
OPCV_PATH=r"F:/soft/c++/opencv"

def findface(image):
    #人脸识别，获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.015,
2, cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))

    result = []
    for r in rect:
```

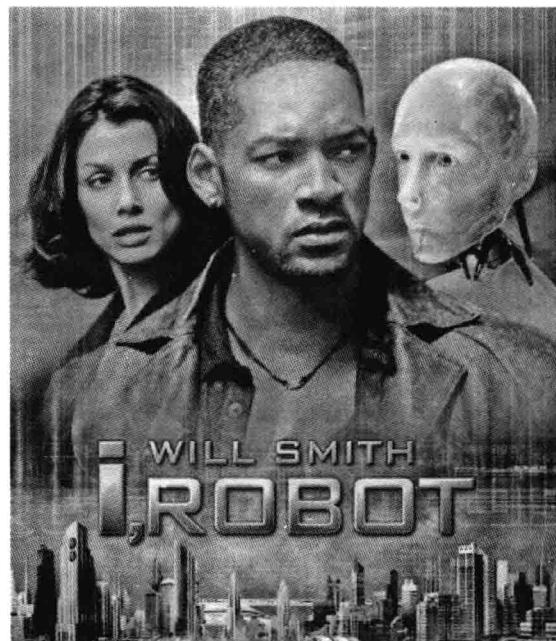


图 9-16 《机械公敌》的电影海报

```

        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result

fn='facesb.png'
my_img=cv.LoadImage(fn)

#获取脸在图像中的坐标
faceresult=findface(my_img)
myimg=cv2.imread(fn)
for ii in xrange(0,len(faceresult)):
    cv2.rectangle(myimg, faceresult
[ii][0], faceresult[ii][1],(0,0,250))

cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

程序运行效果如图 9-17 所示。不但所有人类的脸被定位，而且机器人的脸也被成功找到。

9.4.2 人脸辨识

在讲述本节内容之前，先明确一下本节讲述的算法需要完成的任务：通过某人的一张照片，在他与别人的合影中找到他。

1. 算法描述

完成该任务的人脸辨识算法主要过程是：

- (1) 读取两张图像，生成图像矩阵。
- (2) 以两个图像矩阵为基础，调用 OpenCV 的相关函数完成人脸定位。
- (3) 读取两张图像的人脸区域，生成人脸图像矩阵，并将人脸矩阵转换为灰度图。
- (4) 比较分析人脸图像矩阵，找到最相近的人脸。

2. 欧氏距离算法

在进行人脸识别时，可使用标准欧氏距离算法，该算法不仅简单，而且实用。下面以一张 Microsoft 公司员工与比尔·盖茨的合影和在比尔·盖茨办公室中他与某人的合影为例讲解该算法。

算法基本原理是：将标准欧氏距离算法作为比较分析人脸图像矩阵方法。首先，将两个人脸图像调整为指定大小；接着，用所包含像素的三元色数值组成特征组，然后将特征组映射为高维空间的某个点（在此称之为特征点）；最后，计算两个人脸图像的特征点映射到高维空间后的距离，以欧氏距离最小者为最匹配的人脸。具体步骤如下。

- (1) 调用 OpenCV 相关函数定位人脸。代码如下：

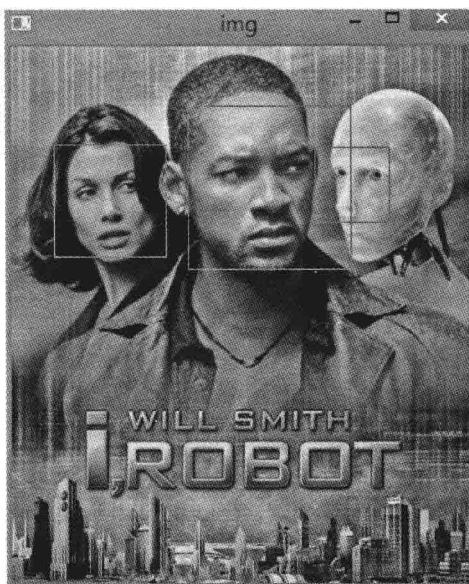


图 9-17 人脸定位

```

cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

(2) 计算欧式距离。代码如下：

```

def get_distance(img, findimg):
    newsize=(img.shape[1],img.shape[0])
    fimg=cv2.resize(findimg,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    return get_EuclideanDistance(my_img,my_fimg)

```

(3) 找到最匹配的人脸。代码如下：

```

myimg=cv2.imread(fn)
myimgt=cv2.imread(fnt)

#IT精英比尔·盖茨
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1], (255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1], (255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1], (255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1], (255,0,255))

```

完整的代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-11.py
#标准欧式距离实现的人脸识别

```

```

import cv2
import numpy as np
import cv2.cv as cv
print 'loading ...'
#请在本程序运行前检查opencv的目录是否为下面的OPCV_PATH值
OPCV_PATH=r"F:/soft/c++/opencv"

def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myv)*(myx-myv)))*np.var(myx-myv)

```

```

def get_distance(img, findimg):
    newsize=(img.shape[1],img.shape[0])
    fimg=cv2.resize(findimg,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    return get_EuclideanDistance(my_img,my_fimg)

def findface(image):
    #人脸识别，获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.1,
2, cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))

    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result

fn='billall1.png'
fnt= 'billtest.png'
my_img=cv.LoadImage(fn)
face_test=cv.LoadImage(fnt)

#获取人脸在图像中的坐标
faceresult=findface(my_img)
facet_result=findface(face_test)

myimg=cv2.imread(fn)
myimgt=cv2.imread(fnt)

#IT精英比尔·盖茨
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0]
[0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1]
[0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1]
[0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1]
[0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.namedWindow('imgt')

```

```

cv2.imshow('imgt', myimgt)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行算法程序后，成功地在员工合影中找到比尔·盖茨。运行本书资源包中的上述代码，可验证人脸识别效果。

3. 改进的欧氏距离算法

前面描述的图像特征码提取算法仅仅是基于像素点的三元色数值的，有时候，图像少量的像素点差异可能干扰识别结果。人脸是一个整体，可能因为人脸的某个部位（如文身、饰品等）、人脸的不同动作和不同角度（如正面人脸、斜侧面人脸以及抬头或低头等）造成少数像素点之间的差异过大，使欧式距离失真（被放大或缩小）。如果从以下两个方面改进上述人脸辩识算法，可以使其识别效果更好。

- 将人脸图像大小设置为适当的数值（通常来说比较小），这样更能突出人脸的特征，而略去很多干扰项。此外，提取原始特征组后，使用PCA降维技术对原始特征组进行进一步加工，生成最终的特征组，从而更好地表征人脸。
- 在标准欧式距离的基础上乘以权重。有两种计算方式：第一，每区域像素设置不同的权重，因为人脸的不同区域表征人脸的能力不同；第二，设置整体权重，使人脸图像矩阵的差分值均匀化。

本节使用第二种方式，加工后的欧式距离不但能更好地表征人脸整体差异，而且不会因为人脸中某些部分过大或过小的差异影响整体识别效果。

在讲解上述算法之前，先讲解一下统计学的变异系数。标准差和方差均可反映数据离散程度，但反映的是离散绝对值。在衡量数据分布离散程度时，不仅要考虑变量值离散程度，还要考虑变量值平均水平。变异系数同时考虑了这两个因素。

变异系数，又称离散系数，是概率分布离散程度的一个归一化量度。它定义为标准差 σ 与平均值 μ 之比：

$$C_v = \frac{\sigma}{\mu}$$

注意，变异系数只在平均值不为零时有定义，人脸差分矩阵正好满足这个条件。

变异系数可以消除因为平均数不同在变异程度比较中产生的干扰。变异系数越小，数据离平均值的偏离程度越小；反之，变异系数越大，数据离平均值的偏离程度越大。

这里首先将变异系数改进，将标准差用方差代替；然后将改进的变异系数的倒数作为计算欧式距离的调节系数（这样做的效果是：将偏离程度较大的数据赋予较小的权重，将偏离程度较小的数据赋予较大的权重中）；最后将标准欧式距离乘以调节权重，从而实现差异平均化，让改进后的欧式矩阵更好地表征人脸整体差异。通过对多个样本的实验来看，这种方式的效果比较理想。

下面以找到威尔·史密斯为例来讲解改进的欧式距离算法。在《机械公敌》中，威尔·史密斯饰演警探戴尔史普纳，布丽姬穆娜饰演苏珊卡尔文博士。以威尔·史密斯出演的

《我是传奇》的电影海报（如图 9-18 所示）中的图像为蓝本，在他与布丽姬穆娜合影的《机械公敌》海报（如图 9-19 和图 9-17 所示）中找到他。

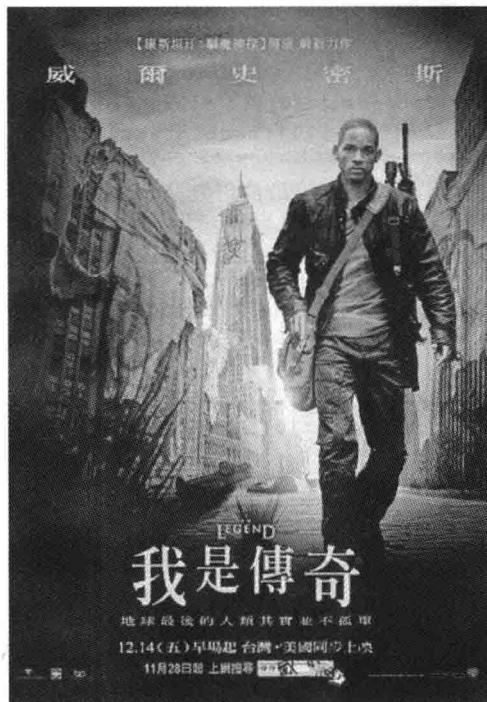


图 9-18 《我是传奇》海报

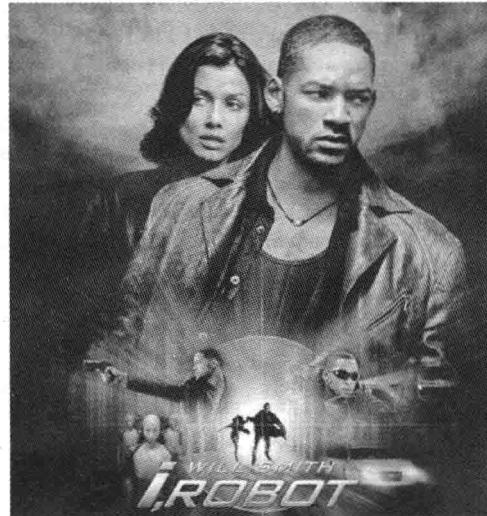


图 9-19 《机械公敌》海报

经过实验，发现标准的欧氏距算法无法完成这个任务，需要使用刚刚描述的改进后的欧氏距离算法。

(1) 在 PCA 降维后，计算基于整体权重的欧氏距离。代码如下：

```
def get_distance(img, findimg):
    newsize=(21,21)
    fimg=cv2.resize(findimg,newsize)
    img=cv2.resize(img,newsize)
    my_img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg, cv2.COLOR_BGR2GRAY)
    pcaimg = mlpy.PCA()
    pcaimg.learn(my_img)
    pca_img = pcaimg.transform(my_img, k=1)
    pca_img=pcaimg.transform_inv(pca_img)
    pcafimg = mlpy.PCA()
    pcafimg.learn(my_fimg)
    pca_fimg = pcaimg.transform(my_fimg, k=1)
    pca_fimg= pcafimg.transform_inv(pca_fimg)
    return get_EuclideanDistance(pca_img,pca_fimg)
```

(2) 根据欧氏距离决定哪个人脸更匹配。代码如下：

```

my_img=cv.LoadImage(fn2)
myimg=cv2.imread(fn2)
#获取脸在图像中的坐标
faceresult=findface(my_img)

#找到威尔·史密斯
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

```

完整的代码如下：

```

#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-12.py
#pcb+改进变异系数人脸识别

import cv2
import numpy as np
import cv2.cv as cv
import mlpy
print 'loading ...'
#请在本程序运行前检查opencv的目录是否为下面的OPCV_PATH值
OPCV_PATH=r"F:/soft/c++/opencv"
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))/(np.var(myx-myv)/abs(np.mean(myx-myv)))

def get_distance(img,findimg):
    newsize=(21,21)
    fimg=cv2.resize(findimg,newsize)
    img=cv2.resize(img,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    pcaimg = mlpy.PCA()
    pcaimg.learn(my_img)
    pca_img = pcaimg.transform(my_img, k=1)
    pca_img=pcaimg.transform_inv(pca_img)

```

```

pcafimg = mlpv.PCA()
pcafimg.learn(my_fimg)
pca_fimg = pcafimg.transform(my_fimg, k=1)
pca_fimg= pcafimg.transform_inv(pca_fimg)
return get_EuclideanDistance(pca_img,pca_fimg)

def findface(image):
    #人脸识别，获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+ "/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.1,
2, cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))
    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])

    return result
fn1='jjgdall.png'
fn2='facesb.png'
fnt='jjgdtest.png'
face_test=cv.LoadImage(fnt)
#获取人脸在图像中的坐标
facet_result=findface(face_test)
myimgt=cv2.imread(fnt)
my_img=cv.LoadImage(fn1)
myimg=cv2.imread(fn1)
#获取人脸在图像中的坐标
faceresult=findface(my_img)
#找到威尔·史密斯
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0]
[0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1]
[0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1]
[0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1]
[0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

cv2.namedWindow('img1')
cv2.imshow('img1', myimg)
my_img=cv.LoadImage(fn2)
myimg=cv2.imread(fn2)
#获取人脸在图像中的坐标
faceresult=findface(my_img)

```

```
#找到威尔·史密斯
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0][0][1]:faceresult[0][1][1],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1], (255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1], (255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1], (255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1], (255,0,255))
cv2.namedWindow('img2')
cv2.imshow('img2', myimg)
cv2.namedWindow('imgt')
cv2.imshow('imgt', myimgt)
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上述程序后，效果良好，在图 9-20 中，以左图中的史密斯的脸为依据，成功地在右边两个图像中找到史密斯（方框中标示了人脸）。



图 9-20 人脸匹配

9.5 手写数字识别

手写数字识别就是指用计算机读取含有手写数字的图像，然后将图像转换为用计算机文字表示的对应数字。SVM 在文本分类、手写文字识别、图像分类、生物序列分析等实际应用中表现出了非常好的性能。下面应用 SVM 算法对 1~9 的手写数字图像进行识别。

9.5.1 手写数字识别算法

(1) 为每个数字各准备 4 个样本图像，如图 9-21 所示。

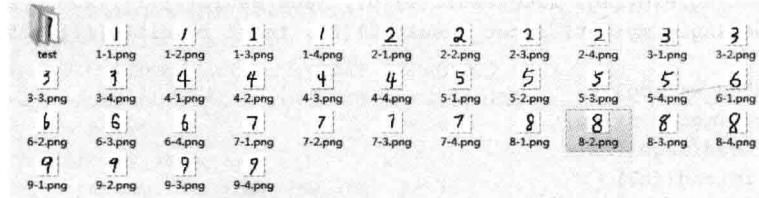


图 9-21 数字样本图像

(2) 将图像大小调整为 8×8 的尺寸, 读取样本图像。

虽然较大的图像较小的图像尺寸更清晰, 但并不意味着它能更好地表现图像的主要特征。不过, 由于手写字体不规范, 因此较大的尺寸会更清楚地表征字体的不规范细节。当图像尺寸调整得更小时, 图像矩阵规模变小, 能表现图像细节的能力减弱, 大部分字体的不规范细节会消失, 但这样一来, 数字的主要特征就更加突出了, 可尽可能消除手写字体不规范带来的影响。比如, 如图 9-22 所示的数字 3, 从左边起第一张是 40×40 的尺寸, 第二张是 20×20 的尺寸, 而第三张则是 8×8 的尺寸。我们在用肉眼观察时, 可以看到, 随着尺寸越来越小, 图像越来越模糊, 不规范的细节也慢慢消失。当计算机使用 8×8 的尺寸来分析数字 3 的图像时, 提取的特征就能更好地表达图像。

在 8×8 (实际是 $8 \times 8 \times 3$, 因为调用 OpenCV 库读取图像后, 会多出一维空间, 多出的这维空间分别放置了红、绿、蓝三元色数值) 的图像矩阵中, 当色彩为黑色 (数字字体的颜色) 时, 表示该像素的特征码为 1, 否则为 0, 形成的像素特征码组成了样本图像特征组。

现在将每个数字类别的样本图像特征组作为输入, 样本对应的数字值作为输出, 用 SVM 进行训练。

(3) 读取未知样本图像, 将未知图像调整为 8×8 的尺寸, 提取图像特征码, 生成图像特征组。

(4) 将未知样本图像特征组送入 SVM 仿真测试, 仿真输出值为根据图像识别出的数字。

9.5.2 算法的 Python 实现

现在根据上述步骤一步步来实现。首先, 提取图像特征, 代码如下:

```
def getnumc(fn):
    '''返回数字特征'''
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(8,8))
    alltz=[]

    for now_h in xrange(0,8):
        xtz=[]
        for now_w in xrange(0,8):
            b = img[now_h,now_w,0]
            g = img[now_h,now_w,1]
            r = img[now_h,now_w,2]
            btz=255-b
            gtz=255-g
            rtz=255-r
            if btz>0 or gtz>0 or rtz>0:
                nowtz=1
            else:
                nowtz=0
            xtz.append(nowtz)
        alltz.append(xtz)
```



图 9-22 不同尺寸的数字 3 图像

```

        xtz.append(nowtz)
        alltz+=xtz
    return alltz

```

然后训练 SVM 并仿真输出。代码如下：

```

x=np.array(x)
y=np.array(y)
svm = mlpv.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=10)
svm.learn(x, y)
print svm.pred(x)

```

完整的 Python 代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-13.py

import numpy as np
import mlpv
import cv2
print 'loading ...'

def getnumc(fn):
    '''返回数字特征'''
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(8,8))
    alltz=[]
    for now_h in xrange(0,8):
        xtz=[]
        for now_w in xrange(0,8):
            b = img[now_h,now_w,0]
            g = img[now_h,now_w,1]
            r = img[now_h,now_w,2]
            btz=255-b
            gtz=255-g
            rtz=255-r
            if btz>0 or gtz>0 or rtz>0:
                nowtz=1
            else:
                nowtz=0
            xtz.append(nowtz)
        alltz+=xtz
    return alltz

#读取样本数字
x=[]
y=[]
for numi in xrange(1,10):
    for numij in xrange(1,5):
        fn='nums/'+str(numi)+'-'+str(numij)+'.png'

```

```

x.append(getnumc(fn))
y.append(numi)

x=np.array(x)
y=np.array(y)
svm = mlpv.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=10)
svm.learn(x, y)
print svm.pred(x)

for iii in xrange(1,10):
    testfn= 'nums/test/' + str(iii) + '-test.png'
    testx=[]
    testx.append(getnumc(testfn))
    print svm.pred(testx)

```

最后，进行仿真测试。

通过对如图 9-23 所示的未知样本的仿真测试得知，运行效果不错。

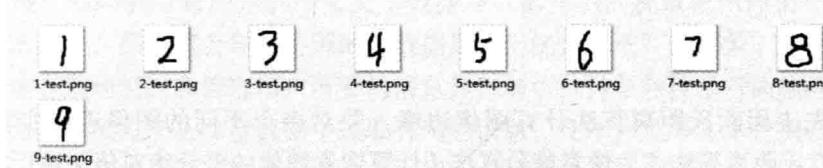


图 9-23 未知数字样本

运行效果如下：

```

loading ...
训练样本测试
[ 1.  1.  1.  1.  2.  2.  2.  3.  3.  3.  3.  4.  4.  4.  4.  5.  5.
 5.  5.  6.  6.  6.  6.  7.  7.  7.  7.  8.  8.  8.  8.  8.  9.  9.  9.  9.]
未知图像测试
nums/test/1-test.png: [ 1.]
nums/test/2-test.png: [ 2.]
nums/test/3-test.png: [ 3.]
nums/test/4-test.png: [ 4.]
nums/test/5-test.png: [ 5.]
nums/test/6-test.png: [ 6.]
nums/test/7-test.png: [ 7.]
nums/test/8-test.png: [ 8.]
nums/test/9-test.png: [ 9.]

```

9.6 小结

使用机器学习算法对数字图像加工的目的是提取所需的知识和信息，主要解决识别与监测、运动分析、场景重建、图像恢复等问题。其算法过程主要包括：图像获取、预处理、特征提取、加工分析、提取知识等。

本章首先介绍数字图像的基础知识，数字图像在计算机中保存为二维整数数组，是二维图像用有限数值像素的表示方式，每个像素对应于二维空间中一个特定的位置；然后通过多个案例的实战，阐述图像识别算法的应用，包括：图像边缘算法、图像匹配算法、人脸辨识算法、图像分类算法、手写数字识别等，并用 Python 编写了代码实现算法，验证了算法的有效性。此外，在讲解算法的同时，剖析算法应用的 SVM、神经网络、PCA 降维、欧氏距离、余弦距离等机器学习技术。

在本章的案例实战中，细心的读者可能会发现，图像样本的质量影响了机器学习算法的效果，SVM 虽然基于小样本计算，但对样本的要求仍然很高，同一类别的样本应保持适当的差异，分别代表这个类别中不同特征的图像，可见，样本的数量和质量都很重要。

本章为讲解方便，使用的样本数量都不多，每个类别的样本数量均在 5 个以下。在实际应用中，用于训练机器学习的样本量应为 100 个以上，对于某些特殊场合的应用，每个类别的样本应在 1000 个以上。

思考题

(1) 首先应用欧氏距离算法计算图像边缘，并对多个不同的图像进行实验，观察效果；然后将欧氏距离算法改为像素差分算法（计算像素数值的差分绝对值）进行实验，观察其效果。

(2) 以多个图像为实验对象，对其进行加噪声点、变形、放大、缩小等操作，应用图像匹配技术进行图像匹配算法实验，尝试在图像匹配算法中应用 PCA 降维技术。

(3) 以多个图像进行更多的人脸辨识实验，尝试应用更多的机器学习算法，比如应用 SVM 和神经网络，或者对欧氏距离算法进行进一步改进，对人脸的不同部位赋予不同的权重，总结各算法的实际应用效果。

(4) 实现从 a 到 z 的 24 个字母图像的识别算法，要求识别未知样本图像表征的字母。

第10章

文本分类案例

在网络信息时代，全球正面临着前所未有的信息爆炸式增长的挑战，中文信息处理也迎来了高速发展。如果拥有着海量的中文文字数据对数据知识的提取依旧停留在过去简单查询、检索的水平上，那么就会导致所谓的“数据爆炸但知识贫乏”的现象。从中文信息中获取知识，快速有效地组织和管理用户所要的信息是当前信息科学和技术所面临的重要挑战。

数据挖掘技术致力于从海量数据中提取有用的信息，文本分类等文本挖掘技术则属于数据挖掘领域中被研究的热点。文本挖掘是人工智能、机器学习、自然语言处理、数据挖掘及相关自动文本处理（如信息抽取、信息检索）等领域中理论和技术结合的产物。文本分类是文本挖掘领域的一个重要分支，而且逐渐成为一个日益重要的研究领域。自动文本分类则会依据文本内容，由计算机根据某种自动分类算法把文本划分为预先定义好的类别，它是对复杂类型的数据进行挖掘的技术。

10.1 文本分类概述

近年来，随着互联网的高速发展和大数据时代的到来，文本分类等文本挖掘技术应用于越来越多的领域。互联网和大数据是信息技术发展催生的一对孪生子。互联网能够方便、准确地记录用户数据，产生了大量的半结构化、非结构化的文本数据，这也使互联网成为大数据分析应用最广泛的领域之一。例如：搜索引擎技术的基础就是基于文本分析算法的，而面向互联网用户的精准营销则是以广告数据分析（现在已经催生了一门科学“计算广告学”）为依据的。海量文档数据库需要高效的文本挖掘算法作为数据索引、查询分析的核心算法，智能输入法需要分析用户输入习惯及输入的词句，自动客服系统由原始的词语匹配技术转变为基于文本挖掘的自然语言理解算法。

分类技术是数据挖掘中非常重要的分支。分类就是根据数据集的特点找出类别的概念描述，这个概念描述代表了这类数据的整体信息，也就是该类的内涵描述，使用这种类的描述可对未来的测试数据进行分类。

文本挖掘算法在搜索引擎中一直扮演着重要的角色，搜索引擎每个阶段的发展都伴随着文本挖掘技术的进步。1990年，作为可搜索FTP文件名列表的 Archie，通过文件名匹配技术完成了文本检索。1993年，世界上第一个 Spider 程序——World Wide Web Wanderer 开始在网络间爬取资料，统计互联网上的服务器数量。同年，Stanford 大学的学生创造了 Excite，它通过分析字词关系进行更有效的检索，文本挖掘技术再次进步。1994年，杨致远和 David Filo 共创办了 Yahoo，Yahoo 网站的分类目录由人工整理维护，他们会精选互联网上的优秀网站，进行简要描述后，分类放置到不同目录下。用户查询时，通过一层层的点击来查找自己想找的网站，人工分类精准度高，但工作量很大。同年 Lycos 正式发布，Lycos 使用了更先进的文本挖掘技术，包括：相关性排序、前缀匹配和字符相近限制、网页自动摘要等。1995年，更多的文本挖掘技术陆续产生，AltaVista 在搜索引擎中引入了自然语言搜索技术，实现高级搜索语法（如 AND、OR、NOT 等）。1997年，文本自动分类技术首次应用于 Northernlight 搜索引擎，该搜索引擎支持对搜索结果进行简单的自动分类。同年，google.com 的域名被 Larry Page 注册，Google 横空出世，之后发展成为今天的搜索业巨头。2013 年全球在线广告营收中，Google 预计占有超过 33% 的市场份额，在全球移动广告营收中，Google 的市场份额约 56%。Google 研究和应用了大量文本挖掘算法，例如：PageRank、动态摘要、网页快照、DailyRefresh、多文档格式支持等。有人评论说 Google 永远改变了搜索引擎的定义。2001年，百度搜索引擎发布，和 Google 一样，百度也研发和使用了很多文本挖掘算法：网页快照、相关搜索词、错别字纠正、Flash 搜索、信息快递搜索、图像搜索、新闻搜索等。

近年来，借助模式识别算法，文本分类技术飞速发展。文本分类大致分为几个要素：文本向量模型表示、文本特征选择和文本训练分类器。目前比较流行的分类方法主要有 SVM、改进余弦相似度、贝叶斯方法、神经网络、k2 最近邻方法、遗传算法、粗糙集等。文本分类算法通常包括文本预处理（中文分词、去除停用词）、文本特征提取、样本特征学习及算法对未知样本的预测等过程。

本章将以余弦相似度、朴素贝叶斯分类算法为主，阐述文本分类技术。

10.2 余弦相似度分类

基于余弦相似度的文本分类算法实现的基本过程为：首先对样本文本进行分词，接着将垃圾词剔除，然后根据剔除后的词条把样本文本中的所有词映射到 n 维空间的一个向量上，并计算未知文本特征组形成的向量与各类别特征组向量之间夹角的余弦值，最后通过比较余弦值的大小判断最接近的分类。

10.2.1 中文分词

中文分词指的是将一个汉字序列切分成一个个单独的词。中文分词是文本挖掘的基础，对于一段中文文本，中文分词是文本自动识别的前提。目前常用的中文分词软件主要有以下几种：

- SCWS。基于词频词典的机械中文分词引擎，它能将一整段的汉字基本正确地切分

成词。采用的是采集的词频词典（词频 TF 指某一个给定的词语在一份给定的文件里出现的次数，近年来，利用计算机进行词频统计，以词频统计的结果来验证词典收词的得失，决定哪些词需要收录，中文词频词典按词频排序存储词语和词组），并辅以一定的专有名称、人名、地名、数字、年代等识别规则，从而达到基本分词的目的。

- ICTCLAS。这是最早的中文开源分词项目，在国内 973 专家组组织的评测活动中获得了第一名，在第一届国际中文处理研究机构 SigHan 组织的评测中获得了多项第一名。ICTCLAS 全部采用 C/C++ 编写，支持 Linux、FreeBSD 及 Windows 系列操作系统，支持 C/C++、C#、Delphi、Java 等主流的开发语言。
- HTTPCWS。基于 HTTP 协议的开源中文分词系统，将取代之前的 PHPCWS 中文分词扩展。
- 庵丁解牛分词。仅支持 Java 语言，且提供 lucence（一款流行的 Java 全文搜索引擎）接口。
- CC-CEDICT。提供一份以汉语拼音为中文辅助的汉英辞典，其词典可以用于中文分词，Chrome 中文版就是使用这个词典进行中文分词的。
- “结巴”(Jieba) 中文分词。Python 中文分词组件 Jieba 支持 3 种分词模式：精确模式，试图将句子最精确地切开，适合文本分析；全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

本章基于 Python 实现算法，因此选择“结巴”中文分词作为分词组件库。下面的代码演示了分词组件 Jieba 的基本使用方法。

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-1.py
import sys
sys.path.append("../")
import jieba

seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print "Full Mode:", "/ ".join(seg_list) # 全模式

seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print "Default Mode:", "/ ".join(seg_list) # 默认模式

seg_list = jieba.cut("他来到了网易杭研大厦")
print ", ".join(seg_list)

seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造") # 搜索引擎模式
print ", ".join(seg_list)
```

演示程序的分词效果如下：

```
/ 我 / 来到 / 北京 / 清华 / 清华大学 / 华大 / 大学 /
Default Mode: 我 / 来到 / 北京 / 清华大学
他， 来到， 了， 网易， 杭研， 大厦
小明， 硕士， 毕业， 于， 中国， 科学， 学院， 科学院， 中国科学院， 计算， 计算所， ，， 后，在， 日本， 京都， 大学， 日本京都大学， 深造
```

中文分词有一个困难，就是会遇到歧义。同样的一句话，可能有两种或者更多的切分方法。主要的歧义有两种：交集型歧义和组合型歧义。例如：语句中出现了“漂亮的”，因为“漂亮”和“亮的”都是词，那么这个短语就可以分成“漂亮 / 的”和“漂 / 亮的”，这种称为交集型歧义。组合型歧义情况更复杂，要根据整个句型来判断，比如：句子“2010年底部队友谊篮球赛结束”，“底部”是一个词，“年底”是一个词，“部队”是一个词，“队友”是一个词，“友谊”是一个词，而分词不能曲解句子含义，这样就产生了分词困难。

目前大部分分词软件能较好地解决歧义问题。下面试着应用“结巴”分词对“2010年底部队友谊篮球赛结束”分词。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-2.py
import jieba
seg_list = jieba.cut("2010年底部队友谊篮球赛结束", cut_all=False)
print "Default Mode:", "/ ".join(seg_list) # 默认模式
```

从下面的执行结果来看，分词效果很理想。“结巴”分词技术比较成熟，能应用于实际工程中。

```
Default Mode:Building Trie..., from E:\WinPython-32bit-2.7.5.1\python-2.7.5\lib\
site-packages\jieba\dict.txt
loading model from cache c:\users\admini~1\appdata\local\temp\jieba.cache
2010/ 年底/ 部队/ 友谊/ 篮球赛/ 结束
loading model cost 1.26200008392 seconds.
Trie has been built succesfully.
```

上面执行结果的第一行中的 Trie 是一种数据结构，“结巴”分词使用它构造词条字典。Trie 称前缀树或字典树，是一种有序树，用于保存关联数组，其中的键通常是字符串。与二叉查找树不同，键不是直接保存在节点中的，而是由节点在树中的位置决定的。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。

10.2.2 停用词清理

停用词又称垃圾词。完成自然语言理解与文本分类等任务时，都需要预处理文本，自动过滤掉某些不能表征意义的字、词或符号，这些字或词被称为停用词（Stop Words）。进行文本分词后形成的词条组中会存在很多停用词，这些词基本不具备表示文本特征的能力，其存在会影响其他词对文本特征的表征能力，因此，需要过滤后才能形成“干净”的文本特征码。

例如：对下面这句话进行分词：“春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高

些快些呢？”代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-3.py

import jieba
seg_list = jieba.cut("春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高些快些呢？", cut_all=False)
print "Default Mode:", "/ ".join(seg_list)##默认模式
.....
.....
```

分词效果如下：

春秋时期/有/一个/农夫/，/他/总是/嫌/田里/的/庄稼/长得/太慢/，/今天/去/瞧瞧/，/明天/去/看看/，/觉得/禾苗/好像/总/没有/长高/。/他/心想/:/有/什么/办法/能/使/它们/长得/高些/快些/呢/?/

上述分词结果中，“有”、“能”、“呢”、“什么”等词以及标点符号都属于停用词的范围，应对其进行清理。清理的方式是建立停用词表，扫描分词结果，从中剔除停用词表中的词条。代码如下：

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-3.py

import jieba
seg_list = jieba.cut("春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高些快些呢？", cut_all=False)
liststr="/ ".join(seg_list)
print u"-----清理前的词条-----"
print "Default Mode:", liststr##默认模式
print u"-----清理后的词条-----"
#停用词清理
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')
for myword in liststr.split('/'):
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        print myword,',',
```

从直观上看，清理停用词后留下的文本词条表征文本的能力都比较强。来比较一下：

-----清理前的词条-----
Default Mode: 春秋时期/有/一个/农夫/，/他/总是/嫌/田里/的/庄稼/长得/太慢/，/今天/去/瞧瞧/，/明天/去/看看/，/觉得/禾苗/好像/总/没有/长高/。/他/心想/:/

有/ 什么/ 办法/ 能/ 使/ 它们/ 长得/ 高些/ 快些/ 呢/ ? /

-----清理后的词条-----

春秋时期，一个，农夫，总是，田里，庄稼，长得，太慢，今天，瞧瞧，明天，看看，觉得，禾苗，好像，长高，心想，办法，长得，高些，快些，

10.2.3 算法实战

1. 任务描述

假设提供了一个描述战争的样本数据，如图 10-1 所示。现在需要对两个未知类型文本进行分析，它们的内容如图 10-2 和图 10-3 所示，判断哪个文本是描述战争类的。

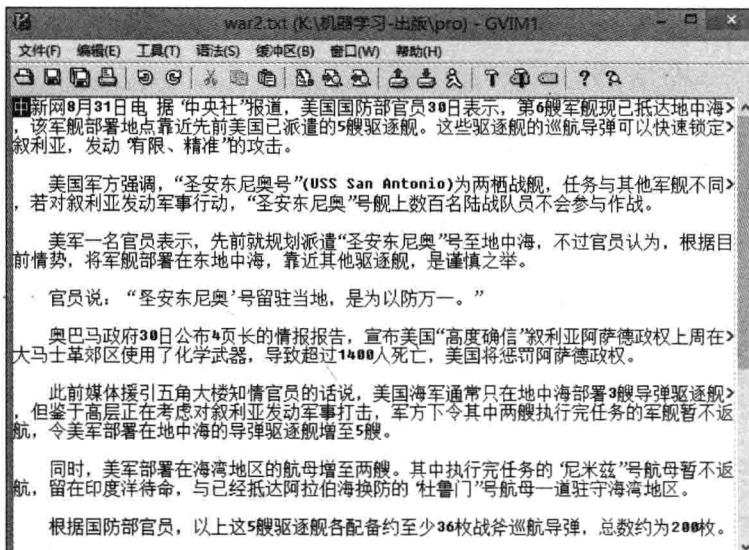


图 10-1 描述战争的样文本本

2. 算法过程

用余弦相似度算法完成上述文本分类任务的过程如下：

- (1) 读取样文本本。
- (2) 对文本进行 utf-8 编码转换。
- (3) 对文本进行预处理，完成中文分词，形成词条库，并去除停用词。
- (4) 读取文本词条库，统计每个词条的词频。词频代表了每个词对一段文本的重要程度，字词的重要性随着它在文件中出现的次数成正比增加。
- (5) 将上一步整理形成的每个词的词频组成文本的词条词频特征码。
- (6) 使用前面第 1 步到第 5 步的方法分析待分类文本，生成待分类文本的词条词频特征码。
- (7) 将待分类文本的词条词频特征码与样本的词条词频特征码进行比较，应用余弦相似度算法判断待分类文本与样本的相似度，取最相似的类型为最终分类的类型。

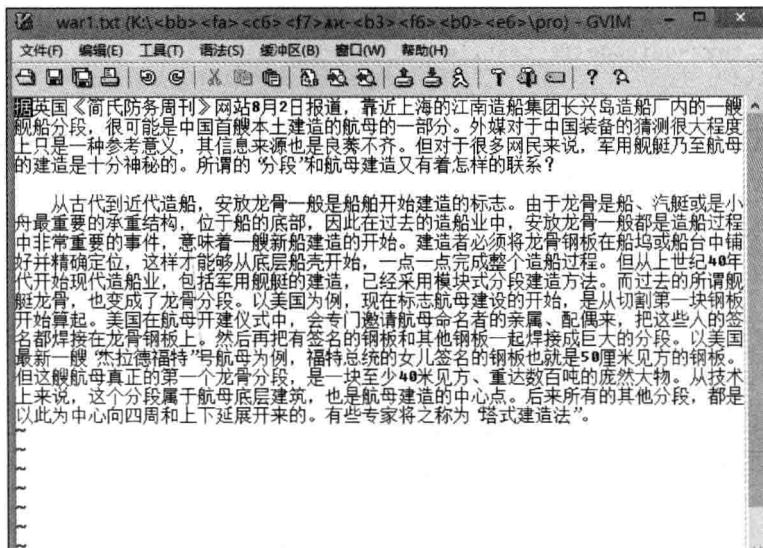


图 10-2 描述战争的未知类型文本

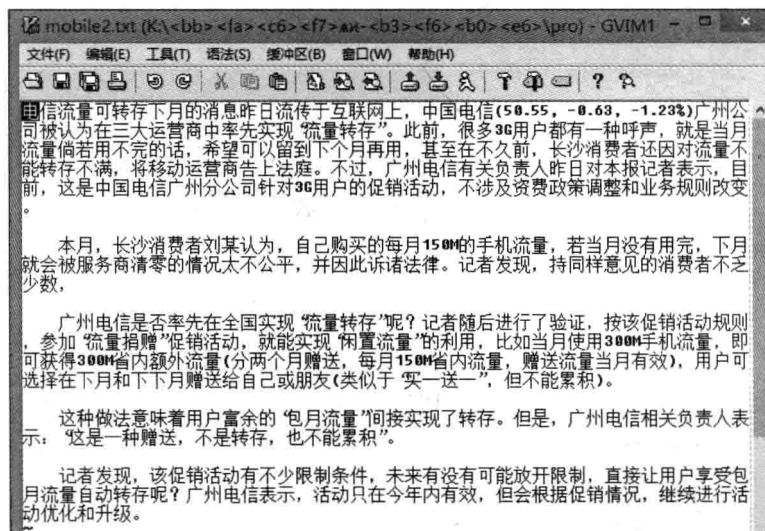


图 10-3 描述手机的未知类型文本

下面用 Python 实现上述算法过程。

(1) 读取样本文本，完成 utf-8 编码转换，然后进行中文分词。

```

print
print 'loading ...'
print 'working',
f1 = open(sampfn)
try:

```

```

f1_text = f1.read()
f1_text=unicode(f1_text,'utf-8')
finally:
    f1.close()
f1_seg_list = jieba.cut(f1_text)

```

(2) 对文本词条进行预处理，去除停用词，计算每个词条的词频。

```

#去除停用词，同时构造样本词的字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

test_words={}
all_words={}
for myword in f1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        test_words.setdefault(myword,0)
        all_words.setdefault(myword,0)
        all_words[myword]+=1

```

(3) 读取待分类文本，进行中文分词。

```

#第一个待测试数据
ftest1 = open(ftest1fn)
try:
    ftest1_text = ftest1.read()
    ftest1_text=unicode(ftest1_text,'utf-8')
finally:
    ftest1.close()
ftest1_seg_list = jieba.cut(ftest1_text)
#第二个待测试数据
ftest2 = open(ftest2fn)
try:
    ftest2_text = ftest2.read()
    ftest2_text=unicode(ftest2_text,'utf-8')
finally:
    ftest2.close()
ftest2_seg_list = jieba.cut(ftest2_text)

```

(4) 继续预处理待分类文本，去除停用词，并生成词频特征码。

```

#读取待测试文本
mytest1_words=copy.deepcopy(test_words)
for myword in ftest1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest1_words.has_key(myword):

```

```

mytest1_words[myword] += 1

mytest2_words = copy.deepcopy(test_words)
for myword in ftest2_seg_list:
    print ".",
    if not (myword.strip() in f_stop_seg_list):
        if mytest2_words.has_key(myword):
            mytest2_words[myword] += 1

```

(5) 计算并输出样本与待测试文本的余弦相似度。

```

#计算样本与待测试文本的余弦相似度
sampdata=[]
test1data=[]
test2data=[]
for key in all_words.keys():
    sampdata.append(all_words[key])
    test1data.append(mytest1_words[key])
    test2data.append(mytest2_words[key])
test1simi=get_cossimi(sampdata,test1data)
test2simi=get_cossimi(sampdata,test2data)
print u"%s与样本[%s]的余弦相似度:%f"%(ftest1fn,sampfn,test1simi)
print u"%s与样本[%s]的余弦相似度:%f"%(ftest2fn,sampfn,test2simi)

```

上面这段代码调用了 `get_cossimi` 函数，这是余弦相似度计算函数。该函数的定义如下：

```

def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)

```

(6) 根据屏幕输出的相似度，预测分类。

两个向量之间的角度余弦值确定两个向量是否大致指向相同的方向。如果两个向量有相同的指向，余弦相似度的值为 1；如果两个向量夹角为 90，余弦相似度的值则为 0。可见，余弦相似度越接近 1，两个文本就越相似。

```

.....
mobile2.txt与样本[war2.txt]的余弦相似度:0.160806
war1.txt与样本[war2.txt]的余弦相似度:0.264215

```

上面是代码的执行结果，分析这个结果可得出结论：mobile2.txt 与 war2.txt 的余弦相似度较小，war1.txt 与 war2.txt 的余弦相似度为 0.264215，更接近 1。因此，应将 war1.txt 文本文件划分为战争类。

以下是全部源代码：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-4.py

```

```

import numpy as np
import jieba
import copy
ftest1fn='mobile2.txt'
ftest2fn='war1.txt'
sampfn='war2.txt'

def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)

if __name__ == '__main__':
    print
    print 'loading ...'
    print 'working',
    f1 = open(sampfn)
    try:
        f1_text = f1.read( )
        f1_text=unicode(f1_text,'utf-8')
    finally:
        f1.close( )
    f1_seg_list = jieba.cut(f1_text)
    #第一个待测试数据

    ftest1 = open(ftest1fn)
    try:
        ftest1_text = ftest1.read( )
        ftest1_text=unicode(ftest1_text,'utf-8')
    finally:
        ftest1.close( )
    ftest1_seg_list = jieba.cut(ftest1_text)
    #第二个待测试数据
    ftest2 = open(ftest2fn)
    try:
        ftest2_text = ftest2.read( )
        ftest2_text=unicode(ftest2_text,'utf-8')
    finally:
        ftest2.close( )
    ftest2_seg_list = jieba.cut(ftest2_text)

    #读取样本文本
    #去除停用词，同时构造样本词的字典
    f_stop = open('stopwords.txt')
    try:
        f_stop_text = f_stop.read( )
        f_stop_text=unicode(f_stop_text,'utf-8')

```

```

    finally:
        f_stop.close( )
f_stop_seg_list=f_stop_text.split('\n')

test_words={}
all_words={}
for myword in f1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        test_words.setdefault(myword,0)
        all_words.setdefault(myword,0)
        all_words[myword]+=1

#读取待测试文本
mytest1_words=copy.deepcopy(test_words)
for myword in ftest1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest1_words.has_key(myword):
            mytest1_words[myword]+=1

mytest2_words=copy.deepcopy(test_words)
for myword in ftest2_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest2_words.has_key(myword):
            mytest2_words[myword]+=1

#计算样本与待测试文本的余弦相似度
sampdata=[]
test1data=[]
test2data=[]
for key in all_words.keys():
    sampdata.append(all_words[key])
    test1data.append(mytest1_words[key])
    test2data.append(mytest2_words[key])
test1simi=get_cossimi(sampdata,test1data)
test2simi=get_cossimi(sampdata,test2data)
print u"%s与样本[%s]的余弦相似度:%f"%(ftest1fn,sampfn,test1simi)
print u"%s与样本[%s]的余弦相似度:%f"%(ftest2fn,sampfn,test2simi)

```

10.3 朴素贝叶斯分类

贝叶斯是一种基于概率的学习算法，其性能可与决策树、神经网络等算法相媲美，是文本挖掘技术的典型代表。它以贝叶斯定理为基础，预测成员关系的可能性，由于其具有坚实的数学理论基础，并且能综合先验信息和数据样本信息，因此成为当前机器学习和数据挖掘的研究热点之一。朴素贝叶斯分类器是目前公认的一种简单有效的概率分类方法，这种分类方法具有非常高的计算效率，在某些应用问题上表现出较好的分类精度，因而被广泛地应用于文本挖掘领域。

10.3.1 算法描述

标准的朴素贝叶斯分类算法的执行过程如下：

(1) 获取样本文本，将样本人工分类整理，并进行标记。

(2) 对每个类别的样本文本进行中文分词。

(3) 去除样本文本中垃圾词条。

(4) 将整理后词条合成样本文本的特征组，分析并计算词条频率信息。例如：假设共有 3 个类别的文本，词条 i 在类别 A、B、C 中出现的次数分别为 $COUNT_i(A)$ 、 $COUNT_i(B)$ 、 $COUNT_i(C)$ ，每个类别的词条总数为 $WORDCOUNT(A)$ 、 $WORDCOUNT(B)$ 、 $WORDCOUNT(C)$ ，那么根据词每个类别的词条总数与词条在每个类别出现的次数就能计算词条频率，计算方式是：词语在每个类别出现的次数除以该类别的总词语数。

比如，某类别的词条总数是 100 个，而词条“冬天”出现了 5 次，那么“冬天”一词在该文件中的词频就是 0.05 (5/100)。

(5) 根据词条频率信息，计算词条在各类别文本的先验概率。词条 i 的各类别先验概率计算公式为：

$$P_i(A) = COUNT_i(A)/WORDCOUNT(A)$$

$$P_i(B) = COUNT_i(B)/WORDCOUNT(B)$$

$$P_i(C) = COUNT_i(C)/WORDCOUNT(C)$$

(6) 读取未知样本，进行中文分词，并去除垃圾词，然后形成样本特征组。

(7) 将未知样本特征词条的先验概率代入朴素贝叶斯公式计算后验算概率，计算得到的最大概率的所属类别即为文本所属类别。

10.3.2 先验概率计算

同一条词条在不同类型的文本中出现的频率通常是不一样的，很多词条只会在某些类别的文本中出现，比如说“微软”、“谷歌”、“医保”、“乔布斯”等词条极少出现在战争类题材的文本中，而“黑莓”、“3G”、“手机”、“电信”等词极少出现在健康类题材的文本中。词条先验概率计算通过提取不同类别中样本的词条，分析其在所有类别中出现的概率，它会以词条为键值，生成词条先验概率哈希数组（在 Python 中称为字典结构），以供后期分类算法使用。

根据朴素贝叶斯的先验概率计算公式来看，在后期计算中，需要计算词条先验概率累乘。如果某词在某类型的样本中从来没有出现，其概率为 0，这样将会使累乘结果变为 0，算法变得毫无意义。在计算先验概率后，在每个词条的先验概率基础上加上一个适当的较小的概率值，防止累乘出现 0。此外，某词在某类型的所有样本中未出现，不代表该词不会出现在该类型的所有文本中，因此，这个较小概率值非常有必要加入先验概率的计算中。

10.3.3 最大后验概率

对未知文本分类时，需要计算后验概率，对于在未知文本中出现的词条，提取其在先

验概率哈希数组中的概率值。然后分别计算不同类型哈希数组中出现的词条的先验概率累乘，从而得到未知文本属于不同类型的后验概率，其中，最大概率所属类别即为未知文本所属类别。

10.3.4 算法实现

这里分别提取数量几乎相同的新闻文本作为样本，这些样本属于汽车、财经、健康、教育、军事类新闻，对样本进行分析。为了验证效果，最后使用未在样本中出现的新闻正文链接进行测试，分析该链接指向的新闻所属类别。

1. 新闻爬取

提取新闻文本的原理与搜索引擎相同。首先，通过类似爬虫的程序对新闻进行爬取，分析新闻主页的新闻正文链接；接着爬取新闻正文网页，清理 HTML 标记；然后形成文本样本，为提高效率，仅在内存中形成文本样本，不在本地硬盘保存；最后，以内存数据为基础，进行下一步分析。相关代码如下：

```
#读取网上新闻搜索目录
txt_class=[]
myclassf1 = open('ClassList.txt')
try:
    myclass_str = myclassf1.read()
    myclass_str=unicode(myclass_str,'gbk')
    myclass_text=myclass_str.split()
    for ii in xrange(0,len(myclass_text),2):
        print ".",
        txt_class.append((myclass_text[ii],myclass_text[ii+1]))
finally:
    myclassf1.close()

links=[]
#分类别爬取网页，生成词条数据
for ci in xrange(0,len(txt_class)):
    print u"\n爬取%s类网页：%s" % (txt_class[ci][0],txt_class[ci][1])
    links.append([])
    pattern = re.compile(r'(.*)?\d+\.shtml')
    purl=txt_class[ci][1]
    page=urllib2.urlopen(purl)
    soup = BeautifulSoup(page)
    for link in soup.find_all('a'):
        mylink=link.get('href')
        match = pattern.match(mylink)
        if match and mylink.find("hd")<0:
            basestr="http://www.chinanews.com"
            if mylink.find("chinanews.com")<0:
                mylink=basestr+mylink
                print mylink
                links[ci].append(mylink)
```

2. 先验概率计算

提取不同类别中样本的词条，分析其在所有类别中出现的概率，生成以词条为键值的先验概率字典变量。此外，根据朴素贝叶斯的先验概率计算公式，在后期计算中，需要计算词条先验概率累乘，所以在每个词条的先验概率基础上加上一个适当的较小的概率值，防止累乘出现0。代码如下：

```
#词条在每个样本中出现的次数
basegl=1e-8
wordybcnt={}
lbcnt=np.zeros(len(yb_txt))
#整理计算词条出现次数
for i in xrange(0,len(yb_txt)):
    for j in xrange(0,len(yb_txt[i])):
        for k in xrange(0,len(yb_txt[i][j])):
            my_word=yb_txt[i][j][k].encode('gbk')
            wordybcnt.setdefault(my_word,np.repeat(0,len(yb_txt)).tolist())
            wordybcnt[my_word][i]+=1
            lbcnt[i]+=1

#计算词条先验概率
print u"\n计算词条概率"
ybgl={}
for my_word in wordybcnt.keys():
    ybgl.setdefault(my_word,np.repeat(0.,len(yb_txt)).tolist())
    for ybii in xrange(0,len(yb_txt)):
        ybgl[my_word][ybii]=basegl+wordybcnt[my_word][ybii]/float(lbcnt[ybii])
    print '.',
```

3. 后验概率计算

读取待分类文本的先验概率字典变量，提取每个词条的先验概率值，然后计算不同类型中出现的词条的先验概率累乘，最后得到待分类文本属于不同类型的后验概率。代码如下：

```
#计算待分类文本后验概率
print u"计算待分类文本后验概率"
testgl=None
wordgl=None
testgl=np.repeat(1.,len(yb_txt))
for myword in ftest_seg_list:
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        myword=myword.encode('gbk')
        for i in xrange(0,len(yb_txt)):
            wordgl=ybgl.get(myword)
            if wordgl:
                if wordgl[i]<>0:
                    testgl[i]*=wordgl[i]
                if np.min(testgl)<1e-50:
                    testgl*=1e20
                if np.max(testgl)>1e100:
```

```
testgl/=float(1e30)
```

运行程序，读取网页 <http://www.chinanews.com/edu/2013/09-17/5296319.shtml> 和 <http://finance.chinanews.com/auto/2013/09-16/5290491.shtml>，以这两个网页为测试对象进行分类。执行结果如下：

```
.....
.....
读取待分类文本
http://www.chinanews.com/edu/2013/09-17/5296319.shtml读取成功。
计算待分类文本后验概率
http://www.chinanews.com/edu/2013/09-17/5296319.shtml
:教育
计算待分类文本后验概率
http://finance.chinanews.com/auto/2013/09-16/5290491.shtml
:汽车
```

从以上执行结果看，两个新闻链接被成功地划分到教育类新闻和汽车类新闻，分类效果良好。

本例中，样本数量较小，在实际应用中，每个类别应准备更多的样本文本。通常来说，一个分类效果较好的朴素贝叶斯算法，其每个类别的样本数量大致有 2000~10000 个。近年来，基于朴素贝叶斯分类的改进算法越来越多，最普遍的是在算法中加入权重的影响，针对某些关键词或中心词加上权重，这种方法法被称为加权朴素贝叶斯算法。

关于加权朴素贝叶斯算法的更多细节，可以查看在《厦门大学学报：自然科学版》2012年第4期上刊登的饶丽丽等的文章《基于特征相关的改进加权朴素贝叶斯分类算法》，也可以在 Google 学术搜索 (<http://scholar.google.com.hk/schhp?hl=zh-CN>) 中以“朴素贝叶斯分类算法”为关键字进行搜索。

以下是完整的 Python 代码：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-5.py
#bayes文本分类
#本程序仅做机器学习研究
#本程序对新闻爬取的工作原理与搜索引擎相同，通过分析链接
#直接搜索新闻，计算词条概率

import numpy as np
import jieba
import urllib2
from bs4 import BeautifulSoup
import re

#读取网上新闻搜索目录
txt_class=[]
myclassf1 = open('ClassList.txt')
```

```

try:
    myclass_str = myclassf1.read()
    myclass_str=unicode(myclass_str,'gbk')
    myclass_text=myclass_str.split()
    for ii in xrange(0,len(myclass_text),2):
        print ".",
        txt_class.append((myclass_text[ii],myclass_text[ii+1]))
finally:
    myclassf1.close()

links=[]
#分类别爬取网页，生成词条数据
for ci in xrange(0,len(txt_class)):
    print u"\n爬取%s类网页:%s" % (txt_class[ci][0],txt_class[ci][1])
    links.append([])
    pattern = re.compile(r'(.*)/\d+\.shtml')
    purl=txt_class[ci][1]
    page=urllib2.urlopen(purl)
    soup = BeautifulSoup(page)
    for link in soup.find_all('a'):
        mylink=link.get('href')
        match = pattern.match(mylink)
        if match and mylink.find("hd")<0:
            basestr="http://www.chinanews.com"
            if mylink.find("chinanews.com")<0:
                mylink=basestr+mylink
                print mylink
                links[ci].append(mylink)

#提取正文内容
ybtxt=[]
print u"\n提取正文内容"
for ci in xrange(0,len(txt_class)):
    ybtxt.append([])
    print ".",
    for mypage in links[ci]:
        my_page=urllib2.urlopen(mypage)
        my_soup = BeautifulSoup(my_page,from_encoding="gb2312")
        my_tt=my_soup.get_text("|", strip=True)
        my_txt=my_tt
        my_fs=u'正文|'
        my_fe1=u'【编辑」
        my_fe2=u'标签」
        zw_start=my_txt.find(my_fs)+8
        last_txt=my_txt[zw_start:len(my_txt)]
        zw_end=last_txt.find(my_fe1)
        if zw_end<0:
            zw_end=last_txt.find(my_fe2)
        page_content=my_txt[zw_start:zw_start+zw_end]
        page_content=page_content.replace(r'_acK({aid:1807,format:0,mode:1,gid:1
,serverbaseurl:"me.afp.chinanews.com/"});','').replace('|','').replace(r'{aid:1805,fo

```

```

rmat:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"},'').replace('cK();','')
page_content=page_content.replace(u'1807: 新闻通发页 大画','').replace(u'标
签: ','').replace(u'评论','').replace(u'正文start编辑姓名start编辑姓名','').replace(u'正
文start','')
    if len(page_content.strip())>0:
        try:
            print my_soup.title.string.encode('gb2312')
            page_content=my_soup.title.string+page_content
        except:
            print "...."
        finally:
            print "-done."
            ybtxt[ci].append(page_content)
#分析正文内容
print u"\n分析正文内容..."

#停用词字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

#分类提取正文词条
print u"\n提取正文词条..."
yb_txt=[]
for ci in xrange(0,len(ybtxt)):
    yb_txt.append([])
    for cj in xrange(0,len(ybtxt[ci])):
        yb_txt[ci].append([])
        my_str = ybtxt[ci][cj]
        my_txt=jieba.cut(my_str)
        for myword in my_txt:
            if not (myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
                yb_txt[ci][cj].append(myword)
    print ".",
#词条在每个样本中出现的次数
basegl=1e-10
wordybcnt={}
lbcnt=np.zeros(len(yb_txt))
#整理计算词条出现次数
for i in xrange(0,len(yb_txt)):
    for j in xrange(0,len(yb_txt[i])):
        for k in xrange(0,len(yb_txt[i][j])):
            my_word=yb_txt[i][j][k].encode('gbk')
            wordybcnt.setdefault(my_word,np.repeat(0,len(yb_txt)).tolist())
            wordybcnt[my_word][i]+=1

```

```

    lbcount[i]+=1

#计算词条先验概率
print u"\n计算词条概率"
ybgl={}

for my_word in wordybcoun.keys():
    ybgl.setdefault(my_word,np.repeat(0.,len(yb_txt)).tolist())
    for ybii in xrange(0,len(yb_txt)):
        ybgl[my_word][ybii]=basegl+wordybcoun[my_word][ybii]/float(lbcount[ybii])
    print '.',

#读取待分类文本
print u"\n读取待分类文本"
ftestlinks=[]
ftestlinks.append(r'http://www.chinanews.com/edu/2013/09-17/5296319.shtml')
ftestlinks.append(r'http://finance.chinanews.com/auto/2013/09-16/5290491.shtml')
for mypage in ftestlinks:
    my_page=urllib2.urlopen(mypage)
    my_soup = BeautifulSoup(my_page,from_encoding="gb2312")
    my_tt=my_soup.get_text("|", strip=True)
    my_txt=my_tt
    my_fs=u'正文|'
    my_fe1=u'[编辑]'
    my_fe2=u'标签'
    zw_start=my_txt.find(my_fs)+8
    last_txt=my_txt[zw_start:len(my_txt)]
    zw_end=last_txt.find(my_fe1)
    if zw_end<0:
        zw_end=last_txt.find(my_fe2)
    page_content=my_txt[zw_start:zw_start+zw_end]
    page_content=page_content.replace(r'_acK({aid:1807,format:0,mode:1,gid:1,serverbseurl:"me.afp.chinanews.com/"})','').replace('!','').replace(r'{aid:1805,format:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"','').replace('cK();','')
    page_content=page_content.replace(u'1807: 新闻通发页 大画','').replace(u'标签: ','').replace(u'评论','').replace(u'正文start编辑姓名start编辑姓名','').replace(u'正文start','')
    page_content=my_soup.title.string+page_content
    print u"%s读取成功.%s" % mypage

#计算待分类文本后验概率
print u"计算待分类文本后验概率"
testgl=None
wordgl=None
testgl=np.repeat(1.,len(yb_txt))
if len(page_content.strip())>0:
    ftest_seg_list = jieba.cut(page_content)
    for myword in ftest_seg_list:
        myword=myword.encode('gbk')
        if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>2:

```

```

for i in xrange(0,len(yb_txt)):
    wordgl=ybgl.get(myword)
    if wordgl:
        if wordgl[i]<>0:
            testgl[i]*=wordgl[i]
        if np.min(testgl)<1e-100:
            testgl*=1e30
        if np.max(testgl)>1e100:
            testgl/=float(1e30)

#计算最大归属概率
maxgl=0.
mychoice=0
for ti in xrange(0,len(yb_txt)):
    if testgl[ti]>maxgl:
        maxgl=testgl[ti]
        mychoice=ti
print "\n\n%s\n:%s"%(mypage,txt_class[mychoice][0])

```

10.4 小结

现在已是数字时代，可以获得和需要处理的文本数量越来越多，这对海量文本数据的分类和处理也提出了更高的要求，文本挖掘技术应运而生。而文本分类作为文本挖掘的主要技术，近年来更是被广泛采用。

文本分类是指依据文本内容，由计算机根据某种自动分类算法，把文本划分为预先定义好的类别。本章基于余弦相似度、朴素贝叶斯分类算法阐述了文本分类及网页分类技术，并用 Python 实现算法。同时详细讲解了文本预处理技术（中文分词、去除垃圾词条）、文本特征提取技术、未知样本分类等算法过程，列举了分类算法应用过程中可能出现的问题以及解决问题的方法。

思考题

- (1) 基于余弦相似度算法对新闻网页进行分类，观察其效果。
- (2) 基于 SVM 算法对新闻网页进行分类，观察其效果。



提示

文本样本的特征组可以定义为其包含的词条在某类别中出现的先验概率。

- (3) 基于加权朴素贝叶斯算法对新闻网页进行分类，观察其效果。



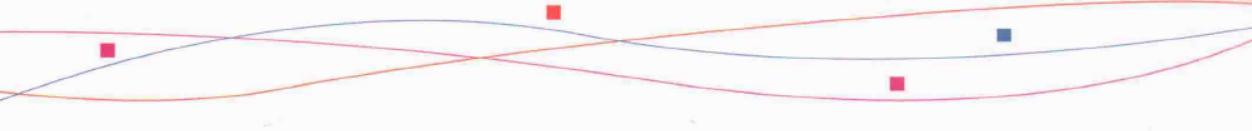
提示

可以以新闻的标题为关键句，将标题中包含的词赋予较大的权重，将权重直接乘以先验

概率作为标题词条的先验概率。

推荐阅读





对于初学者的入门教材而言，本书是一个不错的选择，由浅入深，通俗易懂。本书不在理论上做过多的纠结，讲究“make hands dirty”，在对于机器学习几个主要问题有接触的基础上，引入实践一环，帮助读者能够快速在机器学习的相关职位中上手。从这个方面来说，此书是在快速发展的大数据时代中一本值得

——**李法工程师**

本书从耐心帮助读者了解“什么是机器学习”开始，由浅入深地讲解机器学习相关算法的应用，并附有完整代码，代码能实际运行。这本书适合不同层次的读者，它能帮助读者将相关算法应用于实践工作中。

——**樊恒光 杭州阿里科技有限公司 一搜产品线工程师**

本书是机器学习实践爱好者的盛宴。作者利用计算机上的数学计算工具，循序渐进地展示了与之相关的基本算法及其实现过程。本书的特色在于，书中提供的大量与机器学习应用相关的实践案例，包括统计分析和模式识别的基本方法，都具有较强的代表性，正所谓实践出真知，本书是机器学习实践的宝典，为涉足统计方法和机器学习实践的爱好者指明了一条简单易行的求知之路。

——**宋翼 中国科学院自动化研究所 在读博士**

全书把机器学习的编程语言、数学理论和实例有机结合，是一本值得你拥有的参考书，极具工程价值。书中的朴素贝叶斯分类方法对我来说特别有感触，之前参加过数据挖掘比赛，将此方法应用于推荐系统，具有较高的准确率和鲁棒性等特点。此外，机器学习与嵌入式技术的关系越来越密切，智能物联网正在走近，相信未来的几年会走近千家万户。

——**方家挺 自仪股份 嵌入智能工程师**

机器学习是一门复杂且难以找到突破口的学科，此书结合实际应用，让读者更容易踏入机器学习的大门，参考本书的算法，应用在实际工作中，帮助实现金融智能化。

——**欧阳星 日信证券 软件工程师**

这本书使难懂的机器学习理论变得浅显易懂，书中没有其他教科书上枯燥无味的理论，而是通过实践与应用讲解，让工程师们很快进入数据分析与机器学习领域。

——**张恒 多贝网 运营经理**



投稿热线：(010) 88379604

客服热线：(010) 88378991 88361066

购书热线：(010) 68326294 88379649 68995259 | 华章网站：www.hzbook.com

网上购书：www.china-pub.com

数字阅读：www.henfeidigital.com